A Proposal for an ISO Standard for Public Key Encryption (version 1.1)

Victor Shoup

IBM Zurich Research Lab, Säumerstr. 4, 8803 Rüschlikon, Switzerland

sho@zurich.ibm.com

May 29, 2001

Abstract

This document should be viewed less as a first draft of a standard for public-key encryption, and more as a proposal for what such a draft standard should contain. It is hoped that this proposal will serve as a basis for discussion, from which a consensus for a standard may be formed.

The original version of this document (version 1.0) is dated February 13, 2001.

Version 1.1 contains one substantive change: The decryption algorithm for ACE-Encrypt′ has been slightly modified (see §5.3). Additionally, some minor errors — not affecting the descriptions of any algorithms — have been fixed.

Contents

1 Introduction

1.1 Background

At its meeting on April 3-7 2000 in London, the ISO/IEC JTC 1/SC 27 committee decided to put out a call for contributions for a proposed new project (NP 18033) on *encryption algorithms*. This call for contributions (document SC 27 N 2563) proposed four parts:

1. General
2. Asymmetric Ciphers
3. Block Ciphers
4. Stream Ciphers

The author of this document is currently the acting editor for the Asymmetric Ciphers part of the standard. This document deals exclusively with asymmetric ciphers, a.k.a., public-key encryption schemes.

A number of submissions in response to the call for contributions were received, and are available as ISO document SC 27 N 2656. The author of the present document has carefully reviewed all of the submitted proposals for public-key encryption schemes.

There were a number of different types of schemes submitted:

- Some are based on the hardness of factoring integers or related problems.
- Some are Diffie-Hellman-based schemes — of these, some are based on elliptic curves, and some are based on subgroups of $\mathbf{Z}_p^*$.
- Some allow encryption of arbitrary length messages, and others only allow encryption of short messages.
- Some allow for additional data to be "non-malleably bound" to the ciphertext, while others do not.
- Some allow for cleartexts and ciphertexts to be efficiently processed as "streams," while others do not, requiring more than one pass over this data.
- Some have claims of "provable" security against adaptive chosen ciphertext attack — some relying on the "random oracle" heuristic — some not. For several schemes, these claims of security have proven to be invalid upon closer scrutiny.

Clearly, these submissions are quite incompatible in a number of respects.

1.2 Goals of this document

The goals of this document are as follows:

- To propose a standard functionality that a public-key encryption scheme should implement. This is essentially an *informal* API. Specifying a detailed API is not in the scope of this standard; however, we wish to specify the general structure of such an API — the basic form of the inputs and outputs — and to require that such an API can be efficiently implemented.

- To propose a "unified framework" for Diffie-Hellman-based encryption schemes. This framework unifies both the use of the group and the method of constructing a "hybrid" encryption scheme. In particular:

  – We propose an "abstract group interface" for a group so that any Diffie-Hellman-based encryption scheme can be specified with respect to an abstract group, but yet the group can be implemented in one of several ways, including as a subgroup of an elliptic curve group, and as a subgroup of $\mathbf{Z}_p^*$. The interface is rich enough so as to support all of the subtleties and nuances found in many proposed cryptosystems, especially those using elliptic curves.

  – In order for a Diffie-Hellman-based cryptosystem to be practical, it must be able to process cleartexts that are arbitrary byte strings. There are traditional, and fairly well known "hybrid" schemes to do this: one first uses Diffie-Hellman to derive a shared key, and then encrypts or decrypts the actual payload using symmetric-key techniques. We propose a secure, standard way to do this.

- To propose a set of encryption schemes such that each scheme

  – is "provably" secure against adaptive chosen ciphertext attack in some reasonable sense,

  – conforms to the proposed informal API,

  – conforms to the proposed unified framework for Diffie-Hellman-based encryption (this applies only to Diffie-Hellman-based encryption schemes, of course), and

  – provides a fairly unique efficiency/security tradeoff not provided by other proposed schemes.

In order to achieve the last goal of proposing a set of schemes meeting the stated requirements, we have taken several of the submitted schemes, and proposed modified schemes that meet the stated requirements. Some of these changes are quite minor, while others are more drastic. Some schemes were omitted altogether — given the limited amount of time and other resources available to construct this proposal for a standard, resources had to be concentrated on those schemes which appeared most likely to meet the stated objectives, either with or without minor modification.

Although this document is meant as a proposal for a standard, rather than as a true draft standard, all of the schemes proposed here are specified in great detail, so that going from this proposal to a standard should not necessarily require a great amount of work. Indeed, most of the changes necessary would amount to segregating the normative specifications from the goals and justifications for these specifications, and to harmonizing the notation and terminology with general ISO conventions and with the other parts of the general encryption standard.

1.3 Preliminary remarks on security

Typically, practical symmetric ciphers are designed "from scratch," based partly on established design principles. The security of such a scheme is usually simply taken on faith — there is no justification other than to demonstrate that reasonable design principles were employed in the design of the scheme, and to give (perhaps heuristic) arguments that the scheme resists known types of attacks.

For public-key encryption schemes, the situation is somewhat different. Such a scheme is typically composed of a number of components: besides some kind of "trapdoor" cryptographic transform, there may also be various other components, such as hash functions, symmetric ciphers, etc. Because of this, it is customary nowadays to formally analyze the security of such a scheme relative to the security of its constituent components; that is, to prove the security of the scheme under the assumption that these components satisfy particular, explicit security requirements.

Since proving the security of practical schemes in this way is often infeasible, a heuristic called the *random oracle model* is sometimes used in the proof. In this approach, a cryptographic hash function is modeled — for the purposes of analysis — as a "black box" containing a random function to which the adversary and the algorithms implementing the cryptosystem have "oracle access." This approach has been used implicitly and informally for some time; however, it was formalized by Bellare and Rogaway [BR93], and has subsequently been used quite a bit in the cryptographic research community.

We should stress, however, that the random oracle model is not just "another assumption," like assuming that a hash function is collision resistant, or that a function is pseudorandom. It is a heuristic "leap of faith" — invoking this heuristic is qualitatively a much bigger step than making any particular cryptographic assumption. Indeed, in [CGH98], it is shown that there are cryptosystems that are secure in the random oracle model, but are insecure *no matter what* hash function is used to implement the random oracle.

Despite these problems, the random oracle model is still a useful heuristic and design principle. A proof of security in the random oracle model is still much better than no proof at all, and certainly such a proof does rule out a large family of attacks.

In judging the security of a "provably secure" scheme, there are several independent "dimensions":

- the use or non-use of the random oracle heuristic,
- the "strength" of the underlying assumptions, and

- the efficiency of the security reduction.

Because of these several dimensions, the security of two "provably secure" schemes might be essentially incomparable. For example, one scheme might rely on the random oracle heuristic and a weak assumption, and the other might not use the random oracle heuristic but rely on a stronger assumption, or perhaps the assumptions are simply incomparable.

The efficiency of a security reduction is an issue that is all too often ignored. However, it should be taken into account. For example, a scheme might be secure if RSA inversion is hard, but the security reduction may be so inefficient that for typical sizes of keys — say 1024-bit RSA modulus — the implied algorithm for solving the RSA inversion problem might be slower than the fastest currently known algorithm for factoring numbers.

Even if the security reduction is very inefficient, it can still be argued that such a proof of security nevertheless provides a "qualitative" guarantee of security. Moreover, such a reduction does rule out attacks that would efficiently "scale" to larger sizes of keys.

For public-key encryption schemes, it is widely agreed that the "right" notion of security for a scheme intended for general-purpose use is that of *security against adaptive chosen ciphertext attack*. This notion was introduced in [RS91], and implies other useful properties, like *non-malleability*. See [DDN91, DDN98, BDPR98] for further discussion. In this document, this will be the relevant notion of security used for judging the security of an encryption scheme.

1.4 A summary of submissions and proposed schemes

In this section, we summarize the submissions that were made, give a very brief assessment of of the submissions, and briefly describe the schemes that we actually propose to be included in the standard. It should be stressed that any opinions expressed here, or elsewhere in this document, are solely those of the author of this document.

1.4.1 RSA-OAEP, RSA-OAEP+, and Simple RSA

RSA-OAEP is the fairly well-established RSA encryption scheme, using the padding scheme called OAEP invented by Bellare and Rogaway [BR94], with enhancements and refinements due to Johnson and Matyas [JM96].

The submission coincides with the standards PKCS #1 v2.0 and IEEE P1363.

One of the main supposed virtues of this scheme was an alleged proof in the random oracle model of security against adaptive chosen ciphertext attack, assuming RSA inversion is hard. This "proof" was published in [BR94], and despite years of public scrutiny, it was only recently observed in [Sho00a] that not only is the proof invalid, but that there can be *no* standard proof via "black box" reduction for the OAEP construction in general, given an *arbitrary* one-way trapdoor permutation.

This negative result does not necessarily imply that the specific instance RSA-OAEP is insecure. Indeed, as it turns out — essentially by accident, rather than design — RSA-OAEP is indeed secure in the random oracle model. This was proven for encryption exponent 3 in [Sho00a], and for arbitrary encryption exponent in [FOPS00]. The security reduction is the latter paper is highly inefficient, however.

Another problem with RSA-OAEP is that it only encrypts messages of short length. Indeed, the present author discovered the problem with OAEP while trying to prove the correctness of a modification of OAEP that would allow arbitrary-length messages.

To overcome these problems, we propose in this document a new scheme, called RSA-OAEP+. It is just as efficient as RSA-OAEP, but the general OAEP+ construction is provably secure in the random oracle model (as shown in [Sho00a]). Moreover, even with RSA, the security reduction for OAEP+ is much more efficient than that in [FOPS00] for OAEP, making the scheme more attractive from a concrete security point of view. Also, RSA-OAEP+ is enhanced to deal with arbitrary-length messages.

Even with the security improvements provided by OAEP+, the security reduction is still so inefficient that the security guarantees provided are still not very meaningful in a strict, quantitative sense. For this reason, we also recommend an alternative RSA scheme which is both simpler and more secure, which we call *Simple RSA*. The scheme also fits more nicely into our framework for hybrid encryption.

1.4.2 ECIES

ECIES is the "Elliptic Curve Integrated Encryption Scheme." It is a Diffie-Hellman-based scheme. It is a so-called "hybrid" encryption scheme based on the hardness of the Computational Diffie-Hellman (CDH) problem for elliptic curves. It is closely related to the DHAES construction in [ABR98].

As we shall point out, this scheme is not secure against adaptive chosen ciphertext attack, but can easily be made so with a few small changes. Therefore, we have proposed a variant ECIES′, which besides providing a higher level of security, also has been generalized to conform to our proposed unified framework for Diffie-Hellman-based encryption. The changes between ECIES and ECIES′ are the minimal changes required to ensure security.

The ECIES′ scheme can be proven secure against adaptive chosen ciphertext attack, either by using the rather non-standard assumption in [ABR98], or by using the random oracle heuristic, combined with the (also not very standard) assumption that the CDH problem is hard, even when given access to an oracle for the Decisional Diffie-Hellman (DDH) problem. This latter assumption is called the *gap-CDH* assumption, and is studied in detail in [OP01].

As for efficiency, encryption takes two group exponentiations, and decryption takes one or two (depending on the group, but usually one for elliptic curves).

1.4.3 PSEC

PSEC is a family of Diffie-Hellman-based encryption schemes.

It is claimed that these schemes are all provably secure in the random oracle model, under different assumptions. There are three schemes: PSEC-1, PSEC-2, and PSEC-3.

- For PSEC-1, the DDH problem is assumed to be hard.
- For PSEC-2, the CDH problem is assumed to be hard.
- PSEC-3 is based on the gap-CDH assumption.

We shall argue that actually, these security claims for PSEC-1 and PSEC-2 are unjustified (see §4.4).

We shall propose a scheme PSEC-2′ that is a variant of PSEC-2, and we provide a complete and detailed proof of security in the random oracle model based on the CDH. Besides correcting the security problems of PSEC-2, other changes were made so that the resulting scheme conforms to our proposed API requirements and to our proposed unified framework for Diffie-Hellman-based encryption.

As for efficiency, both PSEC-2′ encryption and decryption require two group exponentiations.

PSEC-1 is based on stronger assumptions, is not significantly more efficient than the other schemes, and has some significant security problems. For these reasons, we have chosen not to include it (or a variant thereof) in this proposal.

PSEC-3 is very similar to ECIES, offering an almost identical efficiency/security tradeoff; since ECIES appears to be the more well-established scheme, we have chosen not to include PSEC-3 (or a variant thereof) in this proposal.

1.4.4 ACE-Encrypt

ACE-Encrypt is a Diffie-Hellman-based encryption scheme that can be proven secure against adaptive chosen ciphertext attack assuming the DDH problem is hard. It is the *only* submission that can truly be proven secure — it does not rely on the random oracle heuristic. It is slightly less efficient than PSEC-2.

The submission is based on the DDH problem for a subgroup of $\mathbf{Z}_p^*$. We have proposed a variant, ACE-Encrypt′. Several changes were made to the original ACE-Encrypt scheme so that the resulting scheme conforms to our proposed API requirements and to our proposed unified framework for Diffie-Hellman-based encryption. However, the resulting scheme is still provably secure — without the random oracle heuristic — based on the DDH, as well as a couple of other reasonable symmetric-key cryptographic assumptions.

As for efficiency, ACE-Encrypt′ encryption requires five group exponentiations, and decryption requires either three or four (depending on the group, but usually three for elliptic curves). Several optimizations are available to reduce the effective costs of these exponentiations, however.

We point out that, like PSEC-2′, ACE-Encrypt′ can be proven secure in the random oracle model under the weaker CDH assumption, although the security reduction for ACE-Encrypt′ is much less efficient than that for PSEC-2′. Additionally, it can be shown that ACE-Encrypt′ is no less secure than ECIES′, in the sense that there is a very tight reduction from an attack on ECIES′ to an attack ACE-Encrypt′. That is, any attack on ACE-Encrypt′ can be converted into an attack on ECIES′, where the running time and success probability of the latter attack are essentially the same as for the former attack. This is discussed in detail in §5.5.2. Thus, any fears that the DDH assumption is "too strong" (see [JN01]) can be safely put to rest.

1.4.5 EPOC

EPOC is a family of encryption schemes based on factoring integers of the form $n = p^2q$.

Security of these schemes is claimed in the random oracle model under one of several assumptions (including the assumption that factoring is hard).

It was the judgment of this author that these schemes should not be included in the standard, for the following reasons:

- the theory on which these schemes are based has not been very widely scrutinized, nor have many of the implementation details;
- they do not seem to offer a particularly attractive efficiency/security tradeoff in relation to the other schemes.

1.4.6 HIME

HIME is a family of encryption schemes based on factoring integers.

Security of these schemes is claimed in the random oracle model under one of several assumptions (including the assumption that factoring is hard).

It was the judgment of this author that these schemes should not be included in the standard. The main reason for this is that the design of the schemes and the claims of security do not appear to stand on very firm ground. Indeed, many details are missing, and it is not at all clear that these gaps can be filled in.

One of these schemes is basically an RSA-OAEP scheme with encryption exponent 2. Given the security problems with OAEP in general (pointed out above), any claims about security of such a scheme must be viewed skeptically.

Indeed, it is possible to design an RSA-OAEP scheme with encryption exponent 2. Not only would such a scheme be very efficient, but its security would be very efficiently reducible to the hardness of factoring integers — a potentially harder problem than the RSA inversion problem.

After the publication of [Sho00a], it has come to light other researchers are currently working on a detailed design of an exponent-2 RSA-OAEP-like scheme. Such a scheme — when its design is complete and publicly scrutinized — would be a good addition to the ISO standard, as it would provide an attractive efficiency/security tradeoff.

1.4.7 Further references on the submissions

The schemes RSA-OAEP, ECIES, PSEC, EPOC, and ACE-Encrypt have also been submitted to the Cryto-Nessie evaluation project, and were presented at the first Crypto-Nessie workshop, held in Leuven on November 13-14, 2000.

Besides the ISO document SC 27 N 2656, detailed descriptions of these algorithms are publicly available at `www.cryptonessie.org/workshop`.

1.4.8 Summary of proposed schemes

So our proposed schemes are as follows:

- Factoring-based schemes:
  - RSA-OAEP+

Scheme	exponentiations per encryption	exponentiations per decryption	random oracle heuristic	main assumption
ECIES′	2	1 (or 2)	yes	gap CDH
PSEC-2′	2	2	yes	CDH
ACE-Encrypt′	5	3 (or 4)	no	DDH

Table 1: Comparison of Diffie-Hellman-based schemes

  - Simple RSA

- Diffie-Hellman-based schemes:
  - ECIES′
  - PSEC-2′
  - ACE-Encrypt′

The reason for submitting two different RSA-based schemes is that they both offer a unique efficiency/security trade-off. While they are both based on the RSA problem, the security reduction for Simple RSA is much more efficient than that for RSA-OAEP+. Additionally, Simple RSA is very simple, and fits more nicely into our general framework for hybrid encryption.

The reason for including three different Diffie-Hellman-based schemes is that they seem to each offer a unique efficiency/security trade-off, as summarized in Table 1.

2 Preliminaries

2.1 Public-key encryption and chosen ciphertext attack

We first define the basic structure of a public-key encryption scheme.

A public-key encryption scheme PKE consists of three algorithms.

- A key generation algorithm $PKE.KeyGen()$, which outputs a public key/secret key pair (PK, SK). The structure of PK and SK depend on the particular scheme.
- An encryption algorithm $PKE.Encrypt(PK, format, L, M)$ that takes as input a public key PK, a *format*, a label L, a cleartext M, and outputs a ciphertext C. Note that L, M, and C are byte strings. The *format* is optional, and its structure and meaning depends on the particular encryption scheme. See Remark 4 below for more on *formats* and Remark 3 below for more on *labels*.
- A decryption algorithm $PKE.Decrypt(SK, L, C)$ that takes as input a secret key SK, a label L, and a ciphertext C, and outputs a cleartext M.

Remark 1 It is important to note cleartexts may be of arbitrary and variable length, although a particular scheme may choose to impose a (very large) upper bound on this length.

Thus, our proposed notion of a public-key encryption scheme is essentially a "digital envelope."

Some currently available public-key encryption schemes, like RSA-OAEP, only allow for very short ciphertexts, and leave it to application engineers to design their own "hybrid" scheme to encrypt long cleartexts (i.e., by encrypting a session key and then using symmetric-key cryptography to encrypt the payload).

However, it seems unrealistic to expect application engineers to correctly design such a secure hybrid scheme. Even PKCS#7 — the standard "digital envelope" mechanism — is not appropriate. The simplest version of this simply encrypts a session key using RSA-OAEP, and then encrypts the message using a standard symmetric cipher — no additional integrity checks are made. Because of this, straightforward application of PKCS#7 yields a trivially *malleable* encryption scheme. $\triangleleft$

Remark 2 Given that cleartexts may be arbitrarily long, a highly desirable property of any public-key encryption scheme should be that both the encryption and decryption algorithms can be efficiently implemented as *filters*. That is, the cleartext may be presented to the encryption algorithm as an input stream, and the ciphertext should be written to an output stream. The algorithm should never have to rewind these streams, and should be able to process these streams using a small, bounded amount of internal storage, independent of the length of these streams. Similarly, the decryption algorithm should be given access to an input stream representing the ciphertext, and the cleartext should be written to an output stream.

Note that in making this requirement, we are not attempting to propose a standard for an encryption API. That would not be in the scope of the proposed standard. Rather, we are simply requiring that the cryptosystems in this standard can efficiently implement such an interface, without specifying any further details of such an interface. $\triangleleft$

Remark 3 A *label* is a byte string that is effectively bound to the ciphertext in a non-malleable way. It may contain data that is implicit from context and need not be encrypted, but that should nevertheless be bound to the ciphertext. We view a label to be a byte string that is meaningful to the application using the encryption scheme, and that is independent of the implementation of the encryption scheme.

For example, there are key exchange protocols in which one party, say A, encrypts a session key K under the public key of the other party, say B. In order for the protocol to be secure, party A's identity (or public key or certificate) must be non-malleably bound to the ciphertext. One way to do this is simply to append this identity to the cleartext. However, this creates an unnecessarily large ciphertext, since A's identity is typically already known to B in the context of such a protocol. A good implementation of the labeling mechanism achieves the same effect, without increasing the size of the ciphertext.

Labels may also be of arbitrary and variable length, but we do not impose the restriction that the encryption and decryption algorithms should be able to process labels as streams.

Both the ECIES and RSA-OAEP submissions include the notion of a label (where it is called an *encoding parameter*), although absolutely no indication was given as to the role or function of a label. Nevertheless, it seems to be a potentially useful feature, and so we include it here. $\triangleleft$

Remark 4 Different *format* values may be used for different encryptions. The choice of *format* specifies a particular way to format a ciphertext. Note that this value is chosen by the sender of the encrypted message. We shall assume that for a given cryptosystem, there are only a small, constant number of formatting choices, and that for a given public key, there is always a default *format* value.

For the cryptosystems proposed here, the only use of this parameter is in elliptic-curve cryptosystems, where the encryptor may choose whether or not to use a "compact" point representation, or perhaps a "hybrid" representation.

Making such an implementation-dependent parameter visible to the API is not such an elegant design choice; however, some applications might require such fine-grained control. In particular, the ECIES submission allowed for this, and so we have incorporated this here. $\triangleleft$

Remark 5 Throughout this document, algorithms will always compute a function on their inputs, except that instead of returning a value, they may **fail**. By convention, if an algorithm **fails**, then unless otherwise specified, an algorithm that invokes that algorithm as a sub-routine also **fails**. Thus, **failing** is analogous to "throwing an exception" in many programming languages. $\triangleleft$

We next recall the definition of security against adaptive chosen ciphertext attack, adapted to deal with labels L and *format* values.

We begin by describing the attack scenario.

First, the key generation algorithm is run, generating the public key and private key for the cryptosystem. The adversary, of course, obtains the public key, but not the private key.

Second, the adversary makes a series of arbitrary queries to a *decryption oracle*. Each query is a label/ciphertext pair (L, C) that is decrypted by the decryption oracle, making use of the private key of the cryptosystem. The resulting decryption is given to the adversary; moreover, if the decryption algorithm **fails**, then this information is given to the adversary, and the attack continues. The adversary is free to construct these label/ciphertext pairs in an arbitrary way—it is certainly *not* required to compute them using the encryption algorithm.

Third, the adversary prepares a label L^*, a *format* value, and two "target" cleartexts M_0, M_1, of equal length and gives these to an *encryption oracle*. The encryption oracle chooses $b \in \{0, 1\}$ at random, encrypts M_b with label L^* using the given *format* value, and gives the resulting "target" ciphertext C^* to the adversary.

Fourth, the adversary continues to submit label/ciphertext pairs (L, C) to the decryption oracle, subject only to the restriction that $(L, C) \neq (L^*, C^*)$.

Just before the adversary terminates, it outputs $\hat{b} \in \{0, 1\}$.

That completes the description of the attack scenario.

For a given adversary A, we define the adversary's *guessing advantage*

$$Advantage_{PKE}(A) = \left|\Pr[\hat{b} = b] - 1/2\right|.$$

Security means that $Advantage_{PKE}(A)$ is "acceptably" small for all adversaries A that run in a "reasonable" amount of time.

Note that in proving an encryption scheme secure, the definition we have given is usually the most convenient. However, in analyzing an encryption scheme in a larger context, a slightly different definition is usually more convenient. In this definition, the attack scenario proceeds just as before. However, instead of measuring the adversary's guessing advantage, we measure its *distinguishing advantage*

$$Advantage'_{PKE}(A) = \left|\Pr\left[\hat{b} = 1 | b = 1\right] - \Pr\left[\hat{b} = 1 | b = 0\right]\right|.$$

It follows directly from the definitions by a trivial calculation that for any adversary A,

$$Advantage'_{PKE}(A) = 2 \cdot Advantage_{PKE}(A).$$

Intuitively, one can think of $Advantage'_{PKE}(A)$ as measuring how differently an adversary behaves in two different attack games: in one game, M_0 is always encrypted, and in the other, M_1 is always encrypted.

In proving analyzing an encryption scheme in a larger context, one usually substitutes an encryption of a secret message by an encryption of a garbage message (all zeros, or random) of the same length, and then analyzes how the adversary behaves. Many secret messages may be replaced by garbage strings, and the distinguishing advantage simply sums (although for some schemes, one can exhibit an even tighter security reduction). A small distinguishing advantage implies that the adversary will not behave significantly differently when this substitution is made. See [BBM00] for more details.

This definition, in slightly different form, was first proposed by Rackoff and Simon [RS91]. It is generally agreed in the cryptographic research community that this is the "right" security property for a general-purpose public-key encryption scheme. This notion of security implies other useful properties, like *non-malleability*. See [DDN91, DDN98, BDPR98] for more on notions of security for public-key encryption schemes.

Remark 6 Note that in the attack game, the adversary is required to submit two target cleartexts of *equal* length. This restriction on the adversary reflects the fact that we cannot expect to hide the length of an encrypted message from the adversary—for many cryptosystems, this will be evident from the length of the ciphertext. It is in general up to the application using the cryptosystem to ensure that the length of a cleartext does not reveal sensitive information. $\triangleleft$

Remark 7 There is a subtle interaction between **failing**, as discussed in Remark 5 and the notion of a stream, discussed Remark 2. An application reading the output stream of the decryption algorithm should take care not to leak any information about the cleartext it has read from that stream, until the decryption process has finished without **failing**. If it does not do this, the application could potentially forfeit the security guarantees of the scheme. $\triangleleft$

Remark 8 Note that none of these definitions make explicit mention of a *security parameter*. Our point of view is concrete—not asymptotic. We assume that a scheme specifies a *particular* security parameter (or set of parameters). If one wants to translate these definitions into ones compatible with the "asymptotic complexity" point of view, then one can consider families of algorithms indexed by a parameter $\lambda = 1, 2, 3, \ldots$ that run in time bounded by a polynomial in λ. Both the scheme and the adversary are viewed as families of algorithms. One can consider either uniform or non-uniform families of algorithms. Security means that the adversary's advantage is "negligible" in λ, meaning that it goes to zero faster than the inverse of any polynomial in λ. $\lhd$

2.2 Key encapsulation

In designing an efficient public-key encryption scheme, a useful approach is to design a "hybrid" scheme, where one uses public key cryptography to encrypt a key that can then be used to encrypt the actual payload using symmetric key cryptography.

To build a hybrid scheme, we need several building blocks. The first is a method for using public key cryptography to "encapsulate" a symmetric key. We call such a scheme a *key encapsulation mechanism.*

Briefly, a key encapsulation mechanism works just like a public-key encryption scheme, except that the encryption algorithm takes no input other than the recipient's public key. Instead, the encryption algorithm generates a pair (K, C_0), where K is a bit string of some specified length, and C_0 is an encryption of K, that is, the decryption algorithm applied to C_0 yields K.

One can always use a public-key encryption scheme for this purpose, generating a random bit string, and then encrypting it under the recipient's public key. However, as we shall see, one can construct a key encapsulation scheme in other, more efficient, ways as well.

Now the details.

A key encapsulation mechanism KEM consists of three algorithms.

- A key generation algorithm $KEM.KeyGen()$, which outputs a public key/secret key pair (PK, SK). The structure of PK and SK depend on the particular scheme.
- An encryption algorithm $KEM.Encrypt(PK, format)$ that takes as input a public key PK and a *format* value, and outputs a key/ciphertext pair (K, C_0).
- A decryption algorithm $KEM.Decrypt(SK, C_0)$ that takes as input a secret key SK and a ciphertext C_0, and outputs a key K.

A key encapsulation mechanism also specifies a positive integer $KEM.OutputKeyLen$ — the length of the key output by $KEM.Encrypt$ and $KEM.Decrypt$.

Additionally, we need the following properties. The set of all possible outputs of the encryption algorithm should be a subset of some an easy-to-recognize, prefix-free language.[1] The prefix-freeness property is needed so that we can parse byte strings from left to right, and efficiently "strip off" a ciphertext. Note that if all ciphertexts have the same length, then the prefix-freeness property is trivially satisfied.

We next define security against adaptive chosen ciphertext attack for a key encapsulation mechanism.

We begin by describing the attack scenario.

[1] A language L is *prefix free* if for any two $x, y \in L$, x is not a proper prefix of y.

First, the key generation algorithm is run, generating the public key and private key for the cryptosystem. The adversary, of course, obtains the public key, but not the private key.

Second, the adversary makes a series of arbitrary queries to a *decryption oracle*. Each query is a ciphertext C_0 that is decrypted by the decryption oracle, making use of the private key of the cryptosystem. The resulting decryption is given to the adversary; moreover, if the decryption algorithm **fails**, then this information is given to the adversary, and the attack continues.

Third, the adversary chooses a *format* value, and invokes an *encryption oracle*. The encryption oracle does the following:

1. Run the encryption algorithm using the given *format* value, obtaining a pair (K^*, C_0^*).
2. Generate a random string $\tilde{K}$ of length $KEM.OutputKeyLen$.
3. Choose $b \in \{0, 1\}$ at random.
4. If $b = 0$, output (K^*, C_0^*); otherwise output $(\tilde{K}, C_0^*)$.

Fourth, the adversary continues to submit ciphertexts C_0 to the decryption oracle, subject only to the restriction that $C_0 \neq C_0^*$.

Just before the adversary terminates, it outputs $\hat{b} \in \{0, 1\}$.

That completes the description of the attack scenario.

For an adversary A, the quantities $Advantage_{KEM}(A)$ and $Advantage'_{KEM}(A)$ are defined in exactly the same way (in terms of b and $\hat{b}$) as $Advantage_{PKE}(A)$ and $Advantage'_{PKE}(A)$ for a public-key encryption scheme. Security means that $Advantage_{KEM}(A)$ is "acceptably" small for all adversaries A that run in a "reasonable" amount of time.

Remark 9 Although one could do so, we have chosen not to incorporate the notion of a *label* in the definition of a key encapsulation mechanism. The reason is that the only application we have of a key encapsulation mechanism in this paper is in the construction of a hybrid encryption scheme, and it is easier to implement labels in the hybrid construction than in the key encapsulation mechanism itself. $\triangleleft$

2.3 Byte string/integer conversions

We simply adopt the functions *OS2IP* and *I2OSP* from the IEEE P1363 standard for conversions between byte (a.k.a., octet) strings and integers.

The function $OS2IP(x)$ takes as input a byte string, and outputs an integer. If $x = x_{l-1} \,\|\, x_{l-2} \,\|\, \cdots \,\|\, x_0$, where each x_i is a byte, then

$$OS2IP(x) = \sum_{i=0}^{l-1} x_i \cdot 256^i.$$

In this formula, each byte x_i is interpreted as a base-256 digit. Note that the left-most byte represents the most-significant digit.

The function *I2OSP* is essentially the inverse of *OS2IP*. The function $I2OSP(m, l)$ takes as input two non-negative integers m and l, and outputs the unique byte string x of length l such that $OS2IP(x) = m$, if such an x exists. Otherwise, the function **fails**. Note that the function fails if and only if $m \geq 256^l$.

2.4 Pseudo-random byte generator

A pseudo-random byte generator *PRBG* is a scheme with the following interface. It defines fixed seed length *PRBG.SeedLen* and a function $PRBG.eval(s, l)$ that takes as input a byte string s of length *PRBG.SeedLen* and an integer $l \geq 0$, and produces as output a byte string of length l.

The assumption we make is that for a random seed s, the output is computationally indistinguishable from a random byte string of the same length.

One recommended way to implement a *PRBG* is to simply use a block cipher in counter mode.

An alternative is to use a block cipher in counter mode, but to output the XOR of consecutive pairs of block cipher outputs. This approach yields a higher level of security when l is very large (see [Luc00]).

2.5 Symmetric key encryption

A symmetric key encryption scheme *SKE* specifies a key length *SKE.KeyLen*, along with encryption and decryption algorithms:

- The encryption algorithm $SKE.Encrypt(k, M)$ takes as input a key k of length *SKE.KeyLen* and a cleartext M. It outputs a ciphertext C_1.
- The decryption algorithm $SKE.Decrypt(k, C_1)$ takes as input a key k of length *SKE.KeyLen* and a ciphertext C_1. It outputs a cleartext M.

We shall need the following security property.

Consider the following attack scenario. The adversary generates two messages (byte strings) M_0, M_1 of equal length. A key k is generated. A bit b is chosen, and M_b is encrypted under key k. The resulting ciphertext C_1 is given to the adversary. The adversary makes a guess $\hat{b}$ at b. The adversary's advantage is defined as $|Pr[\hat{b} = b] - 1/2|$.

For an adversary A that chooses M_0, M_1 of length bounded by l, we denote this advantage by $Advantage_{SKE}(A, l)$.

Security means that the advantage is acceptably small.

Note that one can build a secure symmetric key encryption scheme by using a pseudo-random byte generator (see §2.4) to generate a "one time pad," which is then XORed with the plaintext.

2.6 One-time MAC

A one-time message authentication code *MAC* is a scheme that defines two quantities *MAC.KeyLen* and *MAC.TagLen*, along with a function $MAC.eval(k', T)$ that takes a key

k' of length $MAC.KeyLen$ and an arbitrary byte string T as input, and computes as output a byte string tag of length $MAC.TagLen$. We shall need the following security property.

Consider the following attack scenario. A byte string T^* is chosen by the adversary. A key k' is chosen at random. The MAC is evaluated at T^* with key k', and the output tag^* is given to the adversary. The adversary outputs a pair (T, tag), where T is a byte string with $T \neq T^*$ (and not necessarily of the same length as T^*), and tag is a byte string of length $MAC.TagLen$. The adversary's advantage is defined to be the probability that the MAC on input T with key k' is equal to tag.

For an adversary A that chooses T^* of length bounded by l_1 and T of length bounded by l_2, we denote this advantage by $Advantage_{MAC}(A, l_1, l_2)$.

Security means that this advantage should be acceptably small.

There are a number of acceptable one-time MAC schemes.

Any of the schemes specified by the ISO MAC standard should be acceptable. However, none of these methods are very good from the standpoint of provable security.

2.7 Hybrid encryption

Given a key encapsulation mechanism KEM (see §2.2), a symmetric encryption scheme SKE (see §2.5), and a one-time message authentication code MAC (see §2.6), here is how one can build a hybrid encryption scheme that is secure against chosen ciphertext attack. We require that $KEM.OutputKeyLen = SKE.KeyLen + MAC.KeyLen$.

The key generation algorithm, as well as the public key and private key, are the same as for the key encapsulation scheme.

To encrypt a cleartext M with label L, the key encapsulation algorithm is run, yielding a key K of length $SKE.KeyLen + MAC.KeyLen$, and a ciphertext C_0. We parse K as $K = k \,\|\, k'$, where $|k| = SKE.KeyLen$ and $|k'| = MAC.KeyLen$. We encrypt M using SKE under key k, obtaining its encryption C_1. Then we apply MAC to the byte string string $T = C_1 \,\|\, L \,\|\, I2OSP(|L|, 8)$ using k', obtaining tag. The entire ciphertext is $C = C_0 \,\|\, C_1 \,\|\, tag$.

To decrypt a a ciphertext C with respect to a given label L, we first parse C as $C_0 \,\|\, C_1 \,\|\, tag$, where $|tag| = MAC.TagLen$. This parsing is facilitated by the fact that the set of ciphertexts C_0 are a subset of an easy-to-recognize, prefix-free language. This step may fail, of course, if C is not correctly formatted. We next decrypt C_0 using the decryption algorithm of the key encapsulation scheme, obtaining a key K. This fails if $KEM.Decrypt$ fails. We then parse as $K = k \,\|\, k'$, where $|k| = SKE.KeyLen$ and $|k'| = MAC.KeyLen$. Then we apply the MAC with key k' to the byte string string $T = C_1 \,\|\, L \,\|\, I2OSP(|L|, 8)$, and test whether the resulting tag equals the given tag. If not, we report failure. Otherwise, decrypt C_1 under k, obtaining M. It is possible that $SKE.Decrypt$ fails. Finally, we output M.

Note that for security reasons, if any step in the decryption process fails, the decryption process itself should fail, but the precise cause of the failure should not be indicated.

It is straightforward to show that if the underlying components are secure, then the resulting hybrid encryption scheme $HybridPKE$ is secure against adaptive chosen ciphertext attack. Moreover, the reduction is quite "tight" quantitatively.

Specifically, we have the following:

$$\begin{aligned} \mathit{Advantage}_{\mathit{HybridPKE}}(A) \;\; \leq \;\; & 2 \cdot \mathit{Advantage}_{\mathit{KEM}}(A_1) \; + \\ & \mathit{Advantage}_{\mathit{SKE}}(A_2, l_1) \; + \\ & q_D \cdot \mathit{Advantage}_{\mathit{MAC}}(A_3, l_2, l_3). \end{aligned}$$

Here,

- A_1, A_2, A_3 are adversaries that run in about the same time as A,
- l_1 is a bound on the length of the target cleartext,
- l_2 is a bound on the length of the string T^* corresponding to the target ciphertext,
- l_3 is a bound on the length of the strings T corresponding to ciphertexts submitted to the decryption oracle,
- q_D is a bound on the number of decryption oracle queries,
- $\mathit{Advantage}_{\mathit{KEM}}$ is as defined in §2.2,
- $\mathit{Advantage}_{\mathit{SKE}}$ is as defined in §2.5, and
- $\mathit{Advantage}_{\mathit{MAC}}$ is as defined in §2.6.

The proof of this is an easy exercise.

Remark 10 We continue here the discussion started in Remark 7. In our hybrid construction, there is a single *tag* that is checked at the end of the ciphertext stream. This is the simplest approach, and one that is already seen in practice (as in ECIES). In the ACE Encrypt submission, there was actually a *tag* value inserted every kilobyte or so in the ciphertext stream. The reason for this was so that the decryption algorithm would **fail** as soon as it detected a "bad" ciphertext stream. This would greatly enhance the ability of an application to process the output stream of the decryption algorithm in a stream-like fashion — it would not have to wait until the end of the output stream to detect a "bad" stream. It would not be too difficult to specify such a scheme. This point should perhaps be discussed by the ISO committee. $\lhd$

2.8 Hash functions

We shall assume the availability of a cryptographic hash function. Let *Hash* denote the scheme. Then *Hash.OutputLen* denotes the length of the hash function output, and *Hash.eval* denotes the hash function itself, which maps arbitrary length byte strings to byte strings of length *Hash.OutputLen*.

The invocation of *Hash.Eval* may fail if the input length exceeds some (very large) implementation-defined bound.

In the security analysis, we shall make the following types of assumptions about *Hash*:

- It is *collision resistant*, i.e., it is hard to find two inputs x, y with $x \neq y$ such that $Hash.eval(x) = Hash.eval(y)$.

- It is *second-preimage collision resistant*, i.e., for a given set S of byte strings together with a prescribed probability distribution on S, if $x \in S$ is chosen at random, then it is hard to find $y \in S$ with $x \neq y$ such that $Hash.eval(x) = Hash.eval(y)$. The set S and the probability distribution depend on the application.

- It is a good *entropy-smoothing hash function*, i.e., for a given set S of byte strings together with a prescribed probability distribution on S, then if $x \in S$ is chosen at random, the output $Hash.eval(x)$ is computationally indistinguishable from a random byte string of length $Hash.OutputLen$. Of course, for this assumption to be reasonable, it must be the case that the *entropy* of S is sufficiently high.

- We might also choose to view it as a *random oracle*.

Recommended choices for *Hash* are SHA-1 and RIPEMD-160.

2.9 Mask generation functions

It is convenient to have a mask generating function $MGF(x, l)$ that takes as input a byte string x and an integer $l \geq 0$, and outputs a byte string of length l. The string x is of arbitrary length.

The invocation of *MGF* may fail if the input or output lengths exceed some (very large) implementation-defined bound.

In the security analysis, we will often model *MGF* as a random oracle.

A specific security property that is sometimes desirable for a mask generation function is that it be a good *entropy smoothing function*. That is, the input x is chosen at random from a distribution of byte strings with high entropy, then the output should be computationally indistinguishable from a random byte string of the same length.

Sometimes the notion of a mask generating function is called a *key derivation function (KDF)*.

2.9.1 MGF1

A standard choice for *MGF* is the function *MGF1* defined in the IEEE P1363 standard. This function is parameterized by a hash function *Hash* (see 2.8), and is defined as follows. On input (x, l), the output is the first l bytes of

$$Hash.eval(x \,\|\, I2OSP(0,4)) \,\|\, \cdots \,\|\, Hash.eval(x \,\|\, I2OSP(k-1,4)),$$

where $k = \lceil l / Hash.OutputLen \rceil$.

2.9.2 MGF2

Another standard choice is ANSI X9.63 KDF function (as described in the ECIES submission). This function is the same as *MGF1*, except that the counter runs from 1 to k, rather than from 0 to $k-1$. It also provides for an optional "shared data" argument, which we shall have no use for here. In this document, we shall refer to this function as *MGF2*.

2.9.3 Security critique of MGF1 and MGF2

Of course, if one chooses to model *MGF1* (or *MGF2*) as a random oracle in a security analysis, one is free to do so. There is really not much of a rational basis to argue either for or against such a choice.

However, we do not recommend the use of these functions in applications where one requires the entropy smoothing property discussed above. The only point in this document where this is significant is in the analysis of the variant of the ACE Encrypt scheme discussed in §5, whose security analysis is not based on the random oracle heuristic.

Our reasoning is as follows.

If we were to believe that these were good entropy smoothing functions, this would suggest that the function $F_x(y)$ defined by

$$F_x(y) = \mathit{Hash.eval}(x \,\|\, y)$$

should be a "good" pseudo-random function with key x and input y. However, standard hash functions, like SHA-1, are built using a particular block cipher $P_a(b)$ — with key a and input block b — chained in a standard way. Indeed, suppose that *Hash* is SHA-1 with initial chaining value IV and that x is 512-bits long. So in this case, $P_a(b)$ is a block cipher with a 512-bit key size, and a 160-bit block size. Then

$$F_x(y) = P_y(z) \oplus z, \text{ where, } z = P_x(IV) \oplus IV.$$

Assuming that $P_a(b)$ is a good block cipher, and that x is suitably random, then the value z above should be pseudo-random. Therefore, the security of $F_x(y)$ as a pseudo-random function is equivalent to the security of the function $G_z(y)$ defined by

$$G_z(y) = P_y(z) \oplus z$$

as a pseudo-random function with key z and input y. Is $G_z(y)$ a good pseudo-random function? This is not clear. But certainly, this is a quite unorthodox construction that does not appear to be based on any well-worn or otherwise sound principles.

Because of this perceived potential weakness, we propose two further mask generating functions, *MGF3* and *MGF4*. Either of these can be used in a situation where a random oracle is required. However, these functions seem more reasonable in applications where the entropy smoothing property is required.

2.9.4 MGF3

This function is parameterized by a hash function *Hash* and a padding amount $pamt \geq 4$, and is defined as follows. On input (x, l), the output is the first l bytes of

$$Hash.eval(I2OSP(0, pamt) \,\|\, x) \,\|\, \cdots \,\|\, Hash.eval(I2OSP(k-1, pamt) \,\|\, x),$$

where $k = \lceil l/Hash.OutputLen \rceil$.

Recommended choices for the hash function are SHA-1 or RIPEMD-160. Recommended choices for *pamt* are either 4, or the block size of the underlying hash (64 in the case of SHA-1 or RIPEMD-160).

Based upon the way standard hash functions like SHA-1 or RIPEMD-160 are constructed, it seems like a reasonable assumption is that they are good pseudo-random functions, where we view the text input as the key of the function, and we view the initial vector IV as the input to the function. Typical implementations of these hash functions often do not provide an interface that allows the programmer to choose the IV. However, we get the equivalent effect by setting *pamt* to the block size of the underlying hash.

By setting *pamt* to the block size of the underlying hash function, we are able to give a reasonable justification for the security of *MGF3*. If we set *pamt* to another value, such as 4, this justification is no longer valid. Nevertheless, setting $pamt = 4$ does not seem like a completely unreasonable choice, and certainly the arguments we made above against *MGF1* and *MGF2* no longer apply.

2.9.5 MGF4

This function is parameterized by a hash function *Hash* and a pseudo-random byte generator *PRBG* (see §2.4). It is required that $Hash.OutputLen = PRBG.SeedLen$.

On input (x, l), this function outputs

$$PRBG.eval(Hash.eval(x), l).$$

For the hash function, one can use a standard function like SHA-1 or RIPEMD-160. If *PRBG.SeedLen* is less than 20, then one can simply truncate the output of the hash function.

This function will be a good entropy smoothing function, provided *Hash* is a good entropy smoothing function, and provided *PRBG* is secure as a pseudo-random byte generator.

2.10 Abstract groups

We describe a group as an abstract data type.

A *fully specified* group *Group* is a tuple $(\mathcal{H}, \mathcal{G}, \mu, \nu, \mathcal{E}, \mathcal{D}, \mathcal{E}', \mathcal{D}')$, where:

- $\mathcal{H}$ is a finite abelian group in which all group computations are actually performed. Note that this group need not be cyclic.
- $\mathcal{G}$ is a *cyclic* subgroup of $\mathcal{H}$. This is where the real "action" will normally take place in a cryptographic scheme.

- μ is the order (size) of $\mathcal{G}$, and ν is the index of $\mathcal{G}$ in $\mathcal{H}$, i.e., $\nu = |\mathcal{H}|/\mu$.

  We shall always require that $\mathcal{G}$ contains all elements in $\mathcal{H}$ whose order divides μ. For some cryptographic schemes, we make the stronger requirement that $\gcd(\mu, \nu) = 1$. Note the latter requirement implies the former.

  We will not in general require that μ is prime, although we may impose this restriction in certain cases.

- $\mathcal{E}(\mathbf{a}, \mathit{format})$ is an "encoding" function that maps a group element $\mathbf{a} \in \mathcal{H}$ to a byte string, using the formatting rule specified by *format*.

  We require that the set of all outputs of $\mathcal{E}$ is a subset of some easy-to-recognize, prefix-free language.

- $\mathcal{D}(x)$ is a function that **fails** if x is not a proper encoding; otherwise, it returns a group element $\mathbf{a} \in \mathcal{H}$ and a *format* such that $\mathcal{E}(\mathbf{a}, \mathit{format}) = x$. Of course, it should be the case that the decoding operation inverts the encoding operation.

- $\mathcal{E}'(\mathbf{a})$ is a "partial encoding" function that maps a group element $\mathbf{a} \in \mathcal{H}$ to a byte string.

  We require that the set of all outputs of $\mathcal{E}'$ is a subset of some easy-to-recognize, prefix-free language.

- $\mathcal{D}'(x)$ is a function that either **fails** if x is not a proper partial encoding; otherwise, it returns the set containing all group elements $\mathbf{a} \in \mathcal{H}$ such that $\mathcal{E}'(\mathbf{a}) = x$. We will assume that the size of this set is bounded by a small constant.

All of the above algorithms should have efficient implementations. The function $\mathcal{D}'$ will never be used by any of the schemes, but the existence of this function is necessary to analyze their security.

We of course assume that arithmetic in $\mathcal{H}$ can be carried out efficiently, and that random group elements can be efficiently generated. As a matter of convention, we shall always use additive notation for the group. Also, group elements will be typeset in boldface, and $\mathbf{0}$ denotes the identity element of the group.

This abstraction is meant to be flexible enough to model two important classes of groups: subgroups of $\mathbf{Z}_p^*$, and subgroups of elliptic curves.

2.10.1 Subgroups of $\mathbf{Z}_p^*$

Let p be a prime, and consider the multiplicative group of units modulo p, denoted $\mathbf{Z}_p^*$. Let $\mathcal{H}$ denote this group. Let $\mathcal{G}$ denote any subgroup of $\mathbf{Z}_p^*$. Set $\mu = |\mathcal{G}|$ and $\nu = (p-1)/\mu$. Because $\mathcal{H}$ is itself cyclic, it follows that $\mathcal{G}$ contains all elements of $\mathcal{H}$ whose order divides μ, even if $\gcd(\mu, \nu) \neq 1$. The encoding map $\mathcal{E}$ can be implemented using the function *I2OSP*, where all group elements are encoded as byte strings of length $\lceil \log_{256} p \rceil$. There is no need to pass any additional formatting information to the function $\mathcal{E}$. The map $\mathcal{D}$ can be implemented using *OS2IP*. The function $\mathcal{E}'$ is the same as $\mathcal{E}$, and $\mathcal{D}'$ is the same as $\mathcal{D}$.

2.10.2 Subgroups of Elliptic Curves

Let E be an elliptic curve defined over a finite field $\mathbf{F}_q$. Let $\mathcal{H}$ denote this group. Note that $\mathcal{H}$ is not in general cyclic. Let $\mathcal{G}$ denote a cyclic subgroup, and let μ be its order, and ν be its index in $\mathcal{H}$. The encoding/decoding maps $\mathcal{E}$ and $\mathcal{D}$ can be implemented using the techniques described in IEEE P1363. Note that these encoding techniques allow for a variety of formats: uncompressed, compressed, and hybrid. Thus, a group element need not have a unique encoding. The partial encoding map $\mathcal{E}'$ outputs a fixed length byte string encoding of the x-coordinate of the point. For completeness, we define the partial encoding of the zero point on the curve to be a the all-zero byte string. The partial decoding map $\mathcal{D}'$ converts the given by string back into an element of $\mathbf{F}_q$, and then solves a polynomial equation to find the set of possible y-coordinates (there are at most two).

2.11 Intractability assumptions related to groups

Let

$$Group = (\mathcal{H}, \mathcal{G}, \mu, \nu, \mathcal{E}, \mathcal{D}, \mathcal{E}', \mathcal{D}')$$

as in §2.10.

2.11.1 The Computational Diffie-Hellman Problem

The Computational Diffie-Hellman (CDH) problem for this group is as follows. On input $(\mathbf{g}, x\mathbf{g}, y\mathbf{g})$, where $\mathbf{g}$ is a generator for $\mathcal{G}$, and $x, y \in \{0, \ldots, \mu-1\}$, compute $xy \cdot \mathbf{g}$. We assume the inputs are random, i.e., $\mathbf{g}$ is a randomly chosen generator, and x and y are randomly chosen from the set $\{0, \ldots, \mu - 1\}$.

The CDH assumption is the assumption that this problem is intractable.

Note that in general, it is not feasible to even identify a correct solution to the CDH problem (this is the Decisional Diffie-Hellman problem — see below). In analyzing cryptographic systems, the types of algorithms for solving the CDH that most naturally arise are algorithms that produce a list of candidate solutions to a given instance of the CDH problem. For any algorithm A for the CDH problem that produces a list of length at most l, we let $Advantage_{CDH}(A, l)$ denote the probability that this list contains a correct solution to the input problem instance.

Note that in [Sho97], it is shown how to take an algorithm A with $\epsilon = Advantage_{CDH}(A, l)$, and transform this into an algorithm A' that produces a single output that for all inputs is correct with probability $1 - \delta$. The running time of A' is roughly equal to $O(\epsilon^{-1} \log(1/\delta))$ times that of A, plus the time to perform

$$O(\epsilon^{-1} l \log(1/\delta) \log \mu + (\log \mu)^2)$$

additional group operations.

It is well known that the CDH problem is "random self reducible." There are also other reductions that show that when μ is prime, then an algorithm for solving the CDH problem for any specific base can be converted into an algorithm for any other base. This implies that in a cryptosystem that relies on the CDH, the base $\mathbf{g}$ may be chosen to optimize efficiency without substantially diminishing the security.

2.11.2 The Decisional Diffie-Hellman Problem

The Decisional Diffie-Hellman (DDH) problem is as follows.

We define two distributions.

Distribution $\mathbf{R}$ consists of 4-tuples $(\mathbf{g}, x\mathbf{g}, y\mathbf{g}, z\mathbf{g})$, where $\mathbf{g}$ is a random generator for $\mathcal{G}$, and x, y, z are chosen at random from $\{0, \ldots, \mu - 1\}$. Let $X_{\mathbf{R}}$ denote a random variable sampled from this distribution.

Distribution $\mathbf{D}$ consists of 4-tuples $(\mathbf{g}, x\mathbf{g}, y\mathbf{g}, z\mathbf{g})$, where $\mathbf{g}$ is a random generator for $\mathcal{G}$, x, y are chosen at random from $\{0, \ldots, \mu - 1\}$, and $z = xy \bmod \mu$. Let $X_{\mathbf{D}}$ denote a random variable sampled from this distribution.

The problem is to distinguish these two distributions.

For an algorithm A that outputs either 0 or 1, we define

$$Advantage_{DDH}(A) = |\Pr[A(X_{\mathbf{R}}) = 1] - \Pr[A(X_{\mathbf{D}}) = 1]|.$$

The DDH assumption is that this advantage is negligible for all efficient algorithms.

Note that the DDH assumption is only reasonable when μ has only large prime factors. We shall only make the DDH assumption when μ is prime. Also, when μ is prime, the problem is "random self-reducible" (see [Sta96] and [NR97]).

Unlike the CDH problem, there is no general reduction for the DDH problem from an arbitrary base to a specific base. However, there is a reduction from any specific base to a random base. Therefore, it is recommended that the base $\mathbf{g}$ is chosen at random to maximize security.

See [Bon98] and [NR97] for further discussion of the DDH.

2.11.3 The Gap-CDH Problem

The submitters of the PSEC scheme have proposed a new computational assumption, called the *gap-CDH assumption.* This is the assumption that it is hard to solve the CDH problem, even in the presence of an oracle for solving the DDH problem.

This assumption is not entirely unreasonable, as it it is easily seen that there is no "black box" reduction from the CDH problem to the DDH problem. This can easily be proven in the "black box group" or "generic group" model of [Sho97].

For any algorithm A that makes at most q queries to a DDH oracle, we define $Advantage_{GapCDH}(A, q)$ to the the probability that A solves a random instance of the CDH problem.

See [OP01] for more details about this assumption.

3 A variant of ECIES

We present here an encryption scheme that is a slight variant of ECIES, and also bears many similarities to PSEC-3. What we describe is actually just a key encapsulation mechanism. For reference, let us call this key encapsulation scheme ECIES$'$.

We have to describe the key generation, encryption, and decryption algorithms.

3.1 Key Generation

A fully specified group

$$Group = (\mathcal{H}, \mathcal{G}, \mu, \nu, \mathcal{E}, \mathcal{D}, \mathcal{E}', \mathcal{D}')$$

is chosen, together with a generator $\mathbf{g}$ of the group $\mathcal{G}$.

Two additional parameters need to be chosen, which we call *CofactorMode* and *CheckMode*. Each of these parameters take 0/1 values. These modes are used to deal with security problems that can arise when $\nu > 1$. Here are the rules which should be obeyed in setting these modes.

- If $\nu = 1$, then both of these modes should be 0.
- If $\nu > 1$, both modes can be set to 0, provided $\gcd(\mu, \nu) = 1$ and ν is very small. Note that security in this case degrades by a factor of ν.
- If $\nu > 1$, *CofactorMode* may be set to 1 provided $\gcd(\mu, \nu) = 1$.
- At most one of *CofactorMode* and *CheckMode* should be set to 1.

In addition to *Group*, a mask generation function *MGF* needs to be selected.

Next, a number $x \in \{0, \ldots, \mu - 1\}$ is chosen at random, and the group element $\mathbf{h} = x\mathbf{g}$ is computed.

The public key consists of encodings of *Group*, $\mathbf{g}$, and $\mathbf{h}$, along with the value *CofactorMode* and an indication of the choice of *MGF*. The precise format of this encoding is not specified here. [Editor's note: should it be?]

The private key consists of the public key, together with the number x and the value *CheckMode*.

3.2 Encryption

Recall that for a key encapsulation mechanism *KEM*, the goal is to produce a ciphertext C_0 that is an encryption of a key K, where K is a byte string of length $KeyLen = ECIES'.OutputKeyLen$.

In addition to the recipient's public key, which specifies *Group*, $\mathbf{g}$, $\mathbf{h}$, *CofactorMode*, and *MGF*, the encryption algorithm takes an optional *format* argument. If the encoding function $\mathcal{E}$ supports multiple formats (as in the elliptic curve case), this formatting argument will be passed to the encoding function.

The encryption scheme works as follows.

1. Choose $r \in \{0, \ldots, \mu - 1\}$.
2. If $CofactorMode = 1$, set $r' = r \cdot \nu \bmod \mu$; otherwise, set $r' = r$.
3. Compute $\tilde{\mathbf{g}} = r\mathbf{g}$ and $\tilde{\mathbf{h}} = r'\mathbf{h}$.
4. Output the ciphertext
$$C_0 = \mathcal{E}(\tilde{\mathbf{g}}, format),$$
and the key
$$K = MGF(C_0 \,\|\, \mathcal{E}'(\tilde{\mathbf{h}}), KeyLen).$$

3.3 Decryption

The decryption scheme takes as input a byte string C_0, along with a private key, which specifies *Group*, $\mathbf{g}$, $\mathbf{h}$, *CofactorMode*, *CheckMode*, and x. It runs as follows.

1. Parse C_0, obtaining a group element $\tilde{\mathbf{g}}$. This step fails is C_0 is not a proper encoding of a group element.
2. If $\mathit{CheckMode} = 1$, test if $\mu\tilde{\mathbf{g}} = \mathbf{0}$; if not, then fail.
3. If $\mathit{CofactorMode} = 1$, set $\hat{\mathbf{g}} = \nu\tilde{\mathbf{g}}$; otherwise, set $\hat{\mathbf{g}} = \tilde{\mathbf{g}}$.
4. Compute $\tilde{\mathbf{h}} = x\hat{\mathbf{g}}$.
5. Output the key
$$K = \mathit{MGF}(C_0 \,\|\, \mathcal{E}'(\tilde{\mathbf{h}}), \mathit{KeyLen}).$$

3.4 Security considerations

This scheme can be proved secure against adaptive chosen ciphertext attack in the random oracle model under the gap-CDH assumption (see §2.11.3). Here, we model *MGF* as a random oracle.

Indeed, it is straightforward to show that

$$\mathit{Advantage}_{\mathit{ECIES}'}(A) \leq \mathit{Advantage}_{\mathit{GapCDH}}(A', q_H),$$

where

- A' is an algorithm with access to a DDH oracle whose running time is about the same as that of A,
- q_H is a bound on the number of random oracle queries, and
- $\mathit{Advantage}_{\mathit{GapCDH}}$ is as defined in §2.11.3.

It can also be proved secure under an appropriate "oracle hashing" assumption, as put forward in the DHAES paper [ABR98].

3.5 Comparison to ECIES

If we combine the above key encapsulation scheme with the hybrid construction in §2.7, and then instantiate the group with an elliptic curve, we obtain a scheme very similar to the ECIES scheme. However, there are some differences. All of these differences are justified by two goals: to achieve security against adaptive chosen ciphertext attack, and to provide a scheme that implements the interface for a public-key encryption scheme that we have proposed. See §2.1 for a discussion of both the interface and security issues.

We describe here in detail all of the differences between our proposed scheme and ECIES, together with the justification for these changes.

3.5.1 Hashing C_0

In ECIES, the byte string C_0, i.e., the encoding of $\tilde{\mathbf{g}}$, was not included in the input to the mask generating function, while we have included it here. Our reason for including it is that without it, the scheme does not achieve security against adaptive chosen ciphertext attack.

There are a number of simple examples that illustrate why ECIES does not achieve this level of security. In particular, it is *malleable.* If the group is an elliptic curve, and the partial encoding function $\mathcal{E}'$ encodes only the x-coordinate of a point, then the derived key K is the same if one takes a given ciphertext C_0 encoding a point $\tilde{\mathbf{g}}$ and replaces it with an encoding of $-\tilde{\mathbf{g}}$. Of course, this does not represent a catastrophic failure of the system; it simply illustrates that the definition of adaptive chosen ciphertext security is not met in a strict sense. The very same problem arises if one encodes $\tilde{\mathbf{g}}$ under an alternative format, which is also possible with elliptic curves. A similar problem arises if $\nu > 1$ and $\mathit{CofactorMode} = 1$ — in this case, one could add to $\tilde{\mathbf{g}}$ a non-zero element whose order divides ν, and one obtains yet again a different ciphertext that decrypts to the same thing. A similar problem arises yet again if μ is composite — the ECIES proposal does not allow this, but our proposal does.

We believe that the definition of security should be taken literally, unless there is a *very strong* reason to the contrary. There appears to be very little additional overhead in hashing C_0.

Note that the original DHAES proposal [ABR98], on which ECIES is based, does hash C_0. The main reason for this is that DHAES would be malleable without it if groups with composite order were allowed. Since the ECIES proposal allows only groups with prime order, it would at first appear that this reason goes away. However, ECIES re-introduces the same problem by allowing multiple formats for encoding group elements, by using partial encodings of group elements, and by using the cofactor method of ensuring that things lie in an appropriate subgroup.

Besides solving these problems, by hashing C_0, we get a quantitatively more efficient reduction from the gap-CDH problem. If q_D is the number of decryption requests, and q_H is the number of random oracle requests, then without the hash of C_0, the number of DDH oracle calls that must be made is $q_H \cdot q_D$, whereas with the hash of C_0, this drops to q_H. Hashing C_0 also leads to a much more efficient security reduction in the multi-user/multi-message setting (see [BBM00]).

3.5.2 Variable length cleartexts and labels

It is not clear from the submission if ECIES supports variable length cleartexts or not. It is never mentioned explicitly. It does say that the symmetric key encryption key length is fixed at system set-up time, and only recommends the use of "XOR encryption." That would seem to imply that messages are of fixed length, but that is subject to interpretation. It may be the case that the output length of the mask generation function is set dynamically, according to the length of the message.

If variable length messages were allowed, however, the scheme would not be secure against adaptive chosen ciphertext attacks. The reason is that the MAC is evaluated on the string $C_1 \,\|\, L$.

This is our notation: in the notation of the ECIES submission, C_1 corresponds to EM, and L corresponds to *SharedInfo*$_2$. The precise role and meaning of *SharedInfo*$_2$ is never really described in the ECIES submission, but we assume its meaning is application dependent, and that its role should be the same as that described in Remark 3 in §2.1 of this document.

Suppose that C_1 is a simple XOR encryption of a message M with label L. Suppose c is the last byte of C_1 and M' consists of the first $|M|-1$ bytes of M. Then the same ciphertext decrypts to M' under label $c \,\|\, L$. Thus, the scheme is malleable according to our definition.

In our proposed scheme, we apply the MAC to the string $C_1 \,\|\, L \,\|\, \mathit{I2OSP}(|L|, 8)$. This solves the above problem.

Another point concerning labels where our proposal and the ECIES proposal differ that in addition to the label *SharedInfo*$_2$, there is also a label *SharedInfo*$_1$, which is input to the mask generation function. We could not discern a reason why there are two such labels that are used in two different ways. Since our goal was to propose several encryption schemes implementing a common interface, and we could not see why two labels should be required, we have dropped one of them. Perhaps the submitters could clarify the role of these two labels.

Perhaps one label should be seen as being meaningful to the application using the encryption scheme, and that a second label, perhaps better called a *parameter selection*, could be used to allow an application to pass scheme-specific information. This would actually be very similar to our notion of a *format* value. Perhaps there are other choices that can be made "on the fly," such as the choice of symmetric encryption scheme and MAC functions. Should these choices be "hard wired" in the public key, or should the sender of an encrypted message be able to choose these "on the fly," possibly involving a negotiation with the recipient? The view put forward in this proposal is that for simplicity (and by implication, security), these should be "hardwired" into the public key. That also appears to be the view of the ECIES proposal as well, although this may not be a correct interpretation.

3.5.3 Scheme setup and key deployment

The ECIES submission describes a public-key encryption scheme as consisting in general of four components:

1. Scheme setup
2. Key deployment
3. Encryption operation
4. Decryption operation

The notions of *scheme setup* and *key deployment* together seem to correspond to our notion of *key generation* (see §2.1). However, this is not entirely clear, and the submitters of ECIES may wish to clarify this.

In the *scheme setup* phase

- the *decryptor* selects the static parameters of the cryptosystem (MGF, MAC, SKE, $Group$, and group generator $\mathbf{g}$), and

- the *encryptor* obtains these selections in some (unspecified) authentic manner, performs some standard validations on these selections, and chooses the *format* for encoding group elements.

In the *key deployment* phase

- the *decryptor* selects the remaining components of the private and public keys (x and $\mathbf{h} = x\mathbf{g}$), and

- the *encryptor* obtains the remaining components of the public key in some (unspecified) authentic manner, and performs some standard validations on this data.

After *scheme setup* and *key deployment* are complete, encryption and decryption run as usual.

It is unclear why *key generation* is split into two separate phases, or if our proposal for ECIES′ fails to offer some subtle feature that is available in such a two-phase process. The ECIES submitters are encouraged to comment on this.

4 A variant of PSEC-2

We present here a variant of PSEC-2. This is a key encapsulation scheme that can be combined with the general hybrid method in §2.7 to get a full public-key encryption scheme. While the scheme we present here differs in numerous details from the original PSEC-2, we believe it is similar in spirit to the PSEC-2 submission, preserves the main idea of [FO99] on which it is based, and provides very nearly the same security/efficiency trade-off. Let us refer to our modified version of PSEC-2 as PSEC-2′.

4.1 Key Generation

A fully specified group

$$Group = (\mathcal{H}, \mathcal{G}, \mu, \nu, \mathcal{E}, \mathcal{D}, \mathcal{E}', \mathcal{D}')$$

is chosen, together with a generator $\mathbf{g}$ of the group $\mathcal{G}$.

Additionally, a mask generating function *MGF* (see §2.9) should be selected, along with a positive integer *SeedLen*.

Next, a number $x \in \{0, \ldots, \mu - 1\}$ is chosen at random, and the group element $\mathbf{h} = x\mathbf{g}$ is computed.

The public key consists of encodings of *Group*, $\mathbf{g}$, and $\mathbf{h}$, along with an indication of the choice of *MGF* and the value *SeedLen*. The precise format of this encoding is not specified here. [Editor's note: should it be?]

The private key consists of the public key together with x.

4.2 Encryption

Recall that for a key encapsulation mechanism *KEM*, the goal is to produce a ciphertext C_0 that is an encryption of a key K, where K is a byte string of length $\mathit{KeyLen} = \mathit{PSEC2'.OutputKeyLen}$.

In addition to the recipient's public key, which specifies *Group*, $\mathbf{g}$, $\mathbf{h}$, and *MGF*, the encryption algorithm takes an optional *format* argument. If the encoding function $\mathcal{E}$ supports multiple formats (as in the elliptic curve case), this formatting argument will be passed to the encoding function.

We introduce the following notation. Let $\mathit{I0} = \mathit{I2OSP}(0, 4)$ and $\mathit{I1} = \mathit{I2OSP}(1, 4)$.

The encryption algorithm runs as follows.

1. Choose a random byte string s of length *SeedLen*.
2. Compute
$$t = \mathit{MGF}(\mathit{I0} \,\|\, s, \lceil \log_{256} \mu \rceil + 16 + \mathit{KeyLen}),$$
a byte string of length $\lceil \log_{256} \mu \rceil + 16 + \mathit{KeyLen}$.
3. Parse t as $t = u \,\|\, K$, where $|u| = \lceil \log_{256} \mu \rceil + 16$ and $|K| = \mathit{KeyLen}$.
4. Compute $r = \mathit{OS2IP}(u) \bmod \mu$.
5. Compute $\tilde{\mathbf{g}} = r\mathbf{g}$ and $\tilde{\mathbf{h}} = r\mathbf{h}$.
6. Set $\mathit{EG} = \mathcal{E}(\tilde{\mathbf{g}}, \mathit{format})$ and $\mathit{PEH} = \mathcal{E}'(\tilde{\mathbf{h}})$.
7. Compute
$$v = s \oplus \mathit{MGF}(\mathit{I1} \,\|\, \mathit{EG} \,\|\, \mathit{PEH}, \mathit{SeedLen}).$$
8. Output the key K and the ciphertext $C_0 = \mathit{EG} \,\|\, v$.

4.3 Decryption

The decryption algorithm takes the secret key as well as a ciphertext C_0 as input. It runs as follows.

1. Parse C_0 as $C_0 = \mathit{EG} \,\|\, v$, where *EG* is an encoding a group element $\tilde{\mathbf{g}}$, and v is a byte string of length *SeedLen*. This step may, of course, fail.
2. Compute $\tilde{\mathbf{h}} = x\tilde{\mathbf{g}}$.
3. Set $\mathit{PEH} = \mathcal{E}'(\tilde{\mathbf{h}})$.
4. Compute
$$s = v \oplus \mathit{MGF}(\mathit{I1} \,\|\, \mathit{EG} \,\|\, \mathit{PEH}, \mathit{SeedLen}).$$
5. Compute
$$t = \mathit{MGF}(\mathit{I0} \,\|\, s, \lceil \log_{256} \mu \rceil + 16 + \mathit{KeyLen}),$$
a byte string of length $\lceil \log_{256} \mu \rceil + 16 + \mathit{KeyLen}$.

6. Parse t as $t = u \,\|\, K$, where $|u| = \lceil \log_{256} \mu \rceil + 16$ and $|K| = \mathit{KeyLen}$.

7. Compute $r = \mathit{OS2IP}(u) \bmod \mu$.

8. Compute $\bar{\mathbf{g}} = r\mathbf{g}$.

9. Test if $\bar{\mathbf{g}} = \tilde{\mathbf{g}}$; if not, then fail.

10. Output the key K.

4.4 Changes from PSEC-2

There are a number of substantial differences between PSEC-2′ and the PSEC-2.

First and foremost is the fact that the above scheme is just a key encapsulation mechanism. As we discussed in §2.7, using this we can build a hybrid scheme.

The PSEC-2 submission proposed a different kind of hybrid construction. We would recommend the hybrid construction here above the hybrid construction in the PSEC-2 for three reasons.

1. One of the goals of this document is to consolidate the various submissions, taking the best ideas from all of them, and obtaining a small set of schemes, each of which offers something unique. To that end, it seems like a good idea to use the same hybrid construction for all schemes.

2. The hybrid construction proposed here has a distinct advantage over the hybrid construction proposed in PSEC-2. Namely, it facilitates the implementation of the encryption and decryption algorithms as *filters*, reading their inputs as streams, and writing their outputs as streams, without rewinding, and requiring only a small amount of internal storage. See Remark 2 in §2.1. For the original PSEC-2 construction, this does not seem possible.

3. The hybrid construction proposed here does not rely on random oracles, whereas that in PSEC-2 does. It is easy enough to build a hybrid scheme without random oracles, assuming the underlying key encapsulation mechanism is secure, so it seems worthwhile to do so. In particular, we want to be able to include schemes, like the ACE-Encrypt key encapsulation mechanism, that do not use random oracles in their security analysis.

The only disadvantages of our proposed hybrid construction are that the ciphertexts are slightly longer (an additional MAC *tag* is required), and additional code is required for its implementation (the MAC code). The view put forward in this document is that these disadvantages are outweighed by the advantages of conformity with the other schemes, and of facilitating "streaming." This, of course, may be a point of discussion by the ISO committee.

There are some other differences as well. In our scheme, the value v (in our notation) is computed by masking the seed s with a cryptographic hash

$$\mathit{MGF}(\mathit{I1} \,\|\, \mathit{EG} \,\|\, \mathit{PEH}, \mathit{SeedLen}),$$

whereas in PSEC-2, s is masked directly with PEH — no hash at all. Our scheme thus has potentially more compact ciphertexts than PSEC-2. Also, by including EG in the hash, we deal with the multiple-encoding-format problem (PSEC-2 does not allow multiple encoding formats); this also yields a much more efficient security reduction in the multi-user/multi-message model (see [BBM00]).

A serious criticism of the PSEC-2 scheme as submitted is that there is no detailed proof of the claimed security theorem, either in the submission or elsewhere in the literature. In fact, there is some doubt as to whether the scheme actually is secure under the stated assumptions. The problem is the way the value v (our notation) is computed in PSEC-2. As mentioned above, this is computed as $v = s \oplus PEH$. The only requirement in the scheme is that $SeedLen \leq |PEH|$. However, if $SeedLen < |PEH|$, then the ciphertext contains some of bits of PEH *in the clear*. To prove security of this scheme, then, one would (at least) need to show that one could not compute $\tilde{\mathbf{h}}$ from $\tilde{\mathbf{g}}$ and some of the bits of the partial encoding of $\tilde{\mathbf{h}}$. It would appear that requiring that $SeedLen \geq |PEH|$ solves the problem. Note that the stated requirement that $SeedLen \leq |PEH|$ is apparently not a typographic error, since the examples of PSEC-2 in the appendix of the submission all have $SeedLen < |PEH|$.

A similar, but more severe, criticism applies to the PSEC-1 submission. More specifically, in the PSEC-1 encryption algorithm, the ciphertext contains the XOR of the cleartext with a substring of PEH. There is no way the semantic security of this scheme can be based upon the DDH assumption, since the DDH assumption does *not* imply that the bits of an encoding of a group element are pseudo-random.

Also note that our proposed scheme works with any cyclic group, not just elliptic curve groups, and not just groups of prime order.

We should also mention that the scheme we have proposed here bears some similarities not only to the PSEC-2 submission, but also to a very similar scheme presented in [BLK00].

4.5 Security considerations

Since this proposed scheme differs significantly from PSEC-2 and other schemes in the literature, we sketch a security proof in the random oracle model assuming the CDH (see §2.11.1). Here, we view MGF as a random oracle. Note that all relevant inputs to MGF start with either a "zero word" or a "one word." This effectively gives us two independent random oracles,

$$\begin{aligned} H_0 &: \mathbf{B}^{SeedLen} \rightarrow \mathbf{B}^{\lceil \log_{256} \mu \rceil + 16 + KeyLen}, \\ H_1 &: \mathcal{E}(\mathcal{H}) \times \mathcal{E}'(\mathcal{H}) \rightarrow \mathbf{B}^{SeedLen}. \end{aligned}$$

Here, $\mathbf{B}$ denotes the set of *bytes*. Also, $\mathcal{E}(\mathcal{H})$ denotes the set of all encodings of elements in $\mathcal{H}$, using all formats, and $\mathcal{E}'(\mathcal{H})$ denotes the set of all partial encodings of elements in $\mathcal{H}$. In the security analysis, we shall replace the calls to MGF by appropriate queries to H_0 and H_1.

Consider an adversary A that makes q_D calls to the decryption oracle, q_0 calls to H_0 and q_1 calls to H_1.

Let $\mathbf{G}_0$ be the original attack game, and let S_0 be the event that the adversary correctly guesses the hidden bit b in this game (see §2.2). We shall define a sequence of attack games

$\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_k$. Each of these games should be viewed as operating on the same underlying probability space — only the rules for how certain random variables are computed differ. In each game $\mathbf{G}_i$, $1 \le i \le k$, there will be an event S_i corresponding to S_0. We shall show that for all $1 \le i \le k$, that the difference $|\Pr[S_i] - \Pr[S_{i-1}]|$ is negligible, and moreover, it will be evident that in the last game, $\Pr[S_k] = 1/2$. This will imply that $\mathit{Advantage}_{\mathit{PSEC2'}}(A)$, which is equal to $|\Pr[S_0] - 1/2|$, is negligible.

We adopt the following convention. For an arbitrary ciphertext C_0, we denote by

$$EG, v, \tilde{\mathbf{g}}, \tilde{\mathbf{h}}, \mathit{PEH}, s, t, u, K, r, \bar{\mathbf{g}},$$

the values computed by the decryption algorithm on this ciphertext. Some of these may be undefined if the algorithm would fail before the value was computed. We also denote the target ciphertext C_0^*, and define corresponding values $EG^*, v^*, \tilde{\mathbf{g}}^*, \ldots$.

We classify ciphertexts C_0 submitted to the encryption oracle as follows:

Type I $\tilde{\mathbf{g}} \ne \tilde{\mathbf{g}}^*$;

Type II $EG = EG^*$;

Type III $\tilde{\mathbf{g}} = \tilde{\mathbf{g}}^*$, but $EG \ne EG^*$.

Note that all ciphertexts C_0 submitted to the decryption oracle *before* the encryption oracle has been invoked are classified as Type I. Notice that Type III ciphertexts can arise only if the group supports multiple encoding formats, as in the case of elliptic curves.

Let $\mathcal{S}$ denote the set of points s at which the oracle H_0 has been queried either *(i)* directly by the adversary, or *(ii)* by a Type III decryption oracle invocation. The set $\mathcal{S}$ grows over time, as more queries to H_0 are made. For any byte string s of length *SeedLen*, we define $\rho(s)$ to be the number obtained by taking the first $\lceil \log_{256} \mu \rceil + 16$ of $H_0(s)$, converting to an integer, and reducing mod μ.

The following trivial lemma will streamline our arguments.

Lemma 1 *Let E, E', and F be events defined on a probability space such that* $\Pr[E \wedge \neg F] = \Pr[E' \wedge \neg F]$. *Then we have*

$$|\Pr[E] - \Pr[E']| \le \Pr[F].$$

The proof is a simple calculation, which we omit.

We now define our sequence of games $\mathbf{G}_1, \mathbf{G}_2, \ldots$.

Game $\mathbf{G}_1$. We modify the decryption oracle as follows. If the adversary submits a Type II ciphertext C_0, then in game $\mathbf{G}_1$, we summarily reject C_0, without executing the decryption algorithm at all.

Let F_1 be the event that in game $\mathbf{G}_1$ such a ciphertext is rejected that would not have been rejected under the rules of game $\mathbf{G}_0$. Since these two games proceed identically until F_1 occurs, we have $\Pr[S_0 \wedge \neg F_1] = \Pr[S_1 \wedge \neg F_1]$, and applying Lemma 1 with (S_0, S_1, F_1), we have $|\Pr[S_0] - \Pr[S_1]| \le \Pr[F_1]$.

So it suffices to bound $\Pr[F_1]$. Consider a Type II ciphertext C_0 submitted to the decryption oracle in game $\mathbf{G}_1$. Since $C_0 \neq C_0^*$, we must have $v \neq v^*$, which implies $s \neq s^*$. To accept under the rules of game $\mathbf{G}_0$, we must have $r = r^*$.

To make this happen, the adversary must find an input $s \neq s^*$ to H_0 such that $\rho(s) = r^*$. Thus, $\Pr[F_1] \leq (q_0 + q_D)\mu^{-1}(1 + 2^{-128})$. The factor $(1 + 2^{-128})$ comes from the fact that the value r is not exactly uniformly distributed over $\{0, \ldots, \mu - 1\}$.

So we have

$$|\Pr[S_0] - \Pr[S_1]| \leq (q_0 + q_D)\mu^{-1}(1 + 2^{-128}). \tag{1}$$

Game $\mathbf{G}_2$. In this game, we modify the decryption oracle as follows. Suppose a Type I ciphertext C_0 is submitted, and suppose that $s \notin \mathcal{S}$. Then we summarily reject this ciphertext, without ever proceeding past step 4 of the decryption algorithm.

Note that in this game, Type I and II decryption oracle invocations never evaluate H_0 at points not already in $\mathcal{S}$.

Let F_2 be the event that in game $\mathbf{G}_2$ such a ciphertext is rejected that would not have been rejected under the rules of game $\mathbf{G}_1$. These two games proceed identically until F_2 occurs, and so $\Pr[S_1 \wedge \neg F_2] = \Pr[S_2 \wedge \neg F_2]$, and applying Lemma 1 to (S_1, S_2, F_2), we have $|\Pr[S_1] - \Pr[S_2]| \leq \Pr[F_2]$.

So it suffices to bound $\Pr[F_2]$. Consider a ciphertext C_0 as above is submitted to the decryption oracle in game $\mathbf{G}_2$. On the one hand, if the encryption oracle was previously invoked and $s = s^*$, then under the rules of game $\mathbf{G}_1$, we would certainly reject C_0, since $\tilde{\mathbf{g}} \neq \tilde{\mathbf{g}}^*$. On the other hand, if the decryption oracle was not previously invoked or it was but $s \neq s^*$, then H_0 was never queried at s either by the encryption oracle, the decryption oracle, or the adversary, and so the value r is independent of everything in the adversary's view. It follows that the probability that this ciphertext would not be rejected under the rules of game $\mathbf{G}_1$ is at most $\mu^{-1}(1 + 2^{-128})$.

From this, it follows that $\Pr[F_1] \leq q_D\mu^{-1}(1 + 2^{-128})$, and therefore,

$$|\Pr[S_1] - \Pr[S_2]| \leq q_D\mu^{-1}(1 + 2^{-128}). \tag{2}$$

Game $\mathbf{G}_3$. We make another modification to the decryption oracle. In this new game, we process all Type I ciphertexts C_0 as follows. If $\tilde{\mathbf{g}}$ is not equal to $\rho(s')\mathbf{g}$ for any $s' \in \mathcal{S}$, then we reject without any further processing. Otherwise, if $\tilde{\mathbf{g}} = \rho(s')\mathbf{g}$ for some $s' \in \mathcal{S}$, we compute $\tilde{\mathbf{h}} = \rho(s')\mathbf{h}$, and proceed to decrypt just as in game $\mathbf{G}_2$, but starting with step 3 of the decryption algorithm.

We argue that games $\mathbf{G}_2$ and $\mathbf{G}_3$ are identical.

Consider first a ciphertext for which $\tilde{\mathbf{g}}$ is not equal to $\rho(s')\mathbf{g}$ for any $s' \in \mathcal{S}$. This ciphertext would have anyway been rejected under the rules in game $\mathbf{G}_2$. To see this, let $\tilde{\mathbf{g}} = \hat{r}\mathbf{g}$, where $\hat{r} \in \{0, \ldots, \mu - 1\}$. Now, $\hat{r} \neq \rho(s')$ for any $s' \in \mathcal{S}$. Consider the value s. If $s \in \mathcal{S}$, then we would reject under the rules in game $\mathbf{G}_2$, since the test in step 9 would fail; otherwise, if $s \notin \mathcal{S}$, we would also reject under the rules in game $\mathbf{G}_2$, since the special rejection rule introduced in game $\mathbf{G}_2$ would apply.

Next, consider the case where $\tilde{\mathbf{g}} = \rho(s')\mathbf{g}$ for some $s' \in \mathcal{S}$. It is clear that in this case, decryption proceeds exactly as in game $\mathbf{G}_2$.

So we have

$$\Pr[S_3] = \Pr[S_2]. \tag{3}$$

Game $\mathbf{G}_4$. We modify game $\mathbf{G}_3$ to obtain an equivalent game $\mathbf{G}_4$. This rather technical step is a "bridging" step that will facilitate the analysis of more drastic modifications in game $\mathbf{G}_5$.

In game $\mathbf{G}_4$, we introduce

- a random byte string s^+ of length *SeedLen*,
- a random byte string u^+ of length $\lceil \log_{256} \mu \rceil + 16$,
- a random byte string K^+ of length *KeyLen*, and
- a random *oracle*

$$h^+ : \mathcal{E}(\mathcal{H}) \rightarrow \mathbf{B}^{\mathit{SeedLen}}.$$

Game $\mathbf{G}_4$ is identical to game $\mathbf{G}_3$, except that we apply the following special rules:

R1: In the encryption oracle, we perform the following steps:

1. Set $r^+ = \mathit{OS2IP}(u^+) \bmod \mu$.
2. Compute $\tilde{\mathbf{g}}^* = r^+\mathbf{g}$.
3. Set $EG^* = \mathcal{E}(\tilde{\mathbf{g}}^*, \mathit{format})$.
4. Compute $v^* = s^+ \oplus h^+(EG^*)$.
5. Output the key K^+ and the ciphertext $C_0^* = EG^* \parallel v^*$.

R2: In the decryption oracle, when processing a Type III ciphertext, we use the value $h^+(EG)$ in step 4, instead of $H_1(EG, PEH)$.

R3: Whenever the oracle H_0 is queried — by either the adversary or a Type III decryption oracle — at s^+ we respond with $u^+ \parallel K^+$, instead of $H(s^+)$.

R4: Whenever the oracle H_1 is queried — by either the adversary or a Type I decryption oracle — at a point (EG, PEH), where EG is an encoding of $\tilde{\mathbf{g}}^*$ and PEH is a partial encoding of $x\tilde{\mathbf{g}}^*$, we respond with $h^+(EG)$ instead of $H_1(EG, PEH)$.

It is clear that games $\mathbf{G}_3$ and $\mathbf{G}_4$ are completely equivalent, since we have consistently replaced one set of random variables by another set of identically distributed random variables. In particular,

$$\Pr[S_3] = \Pr[S_4]. \tag{4}$$

Game $\mathbf{G}_5$. Game $\mathbf{G}_5$ is the same as game $\mathbf{G}_4$, except that we drop rules **R3** and **R4**, while retaining **R1** and **R2**.

Note that in this game, we do not use the secret key of the cryptosystem at all. Also note that the ciphertext C_0^* is no longer a valid ciphertext in general, nor does it hold in general that $s^* = s^+$, or that $t^* = u^+ \parallel K^+$, since the random oracles are no longer consistent with the modifications made in the encryption oracle. Indeed, K^+ and hence the hidden bit b are independent of the adversary's view in game $\mathbf{G}_5$. The string s^+ is also independent of the adversary's view. Further, the behavior of Type III decryption oracle queries are also not consistent with the random oracles.

Despite these differences, however, games $\mathbf{G}_4$ and $\mathbf{G}_5$ proceed identically until the string s^+ appears in $\mathcal{S}$ or either the adversary or a Type I decryption oracle invocation queries H_1 on inputs (EG, PEH), where EG is an encoding of $\tilde{\mathbf{g}}^*$ and PEH is the partial encoding of $x\tilde{\mathbf{g}}^*$.

Let F_{5a} be the event that in game $\mathbf{G}_5$, the string s^+ appears in $\mathcal{S}$ at some point in time. Let F_{5b} be the event that either the adversary or a Type I decryption oracle invocation queries H_1 on inputs (EG, PEH), where EG is an encoding of $\tilde{\mathbf{g}}^*$ and PEH is the partial encoding of $x\tilde{\mathbf{g}}^*$. Let $F_5 = F_{5a} \vee F_{5b}$.

Since games $\mathbf{G}_4$ and $\mathbf{G}_5$ proceed identically until the point where F_5 occurs, we have $\Pr[S_4 \wedge \neg F_5] = \Pr[S_5 \wedge \neg F_5]$. Applying Lemma 1 with (S_4, S_5, F_5), we have $|\Pr[S_4] - \Pr[S_5]| \leq \Pr[F_5]$.

Since s^+ is independent of the adversary's view, we have

$$\Pr[F_{5a}] \leq (q_0 + q_D)2^{-\mathit{SeedLen}}.$$

Now, $\Pr[F_{5b}]$ is bounded by $(1 + 2^{-128})$ times the probability that an adversary A' — running in expected time nearly the same as the running time of the original adversary A — can construct a list of $O(q_1 + q_D)$ group elements, one of which contains a solution to a given instance of the CDH problem.

This algorithm runs by taking a random instance $(\mathbf{g}, \mathbf{h}, \tilde{\mathbf{g}}^+)$ of the CDH problem as input, and runs A against a slightly modified version of game $\mathbf{G}_5$. In this modified game, we use the given values $\mathbf{g}, \mathbf{h}$ to form the public key in game $\mathbf{G}_5$, Also, we use the given value $\tilde{\mathbf{g}}^+$, instead of deriving it from u^+ (note that u^+ is not used anywhere else in game $\mathbf{G}_5$). Finally, to implement this algorithm, we simulate the random oracles in the usual way, using standard hash table techniques. We also use standard hash table techniques to implement the Type I decryption oracle queries, as modified in game $\mathbf{G}_3$. The factor $(1 + 2^{-128})$ comes from the fact that the distribution of $\tilde{\mathbf{g}}+$ in game $\mathbf{G}_5$ is slightly non-uniform, whereas we assume the corresponding value in the CDH instance is uniformly distributed.

From this, it follows that

$$\begin{aligned} |\Pr[S_4] - \Pr[S_5]| \;\leq\; & \mathit{Advantage}_{\mathit{CDH}}(A', O(q_1 + q_D))(1 + 2^{-128}) + \\ & (q_0 + q_D)2^{-\mathit{SeedLen}}, \end{aligned} \tag{5}$$

where $\mathit{Advantage}_{\mathit{CDH}}$ is as defined in §2.11.1.

It is also clear that in game $\mathbf{G}_5$, the hidden bit b is independent of all values directly or indirectly accessible to the adversary. Hence,

$$\Pr[S_5] = 1/2. \tag{6}$$

Putting together (1), (2), (3), (4), (5), (6), we obtain

$$\begin{aligned} \mathit{Advantage}_{\mathit{PSEC2'}}(A) \;\leq\; & (q_0 + 2q_D)\mu^{-1}(1 + 2^{-128}) + \\ & \mathit{Advantage}_{\mathit{CDH}}(A', O(q_1 + q_D))(1 + 2^{-128}) + \\ & (q_0 + q_D)2^{-\mathit{SeedLen}}. \end{aligned} \tag{7}$$

4.6 Further remarks

Note that in this scheme, we do not have to make an additional check to ensure that $\tilde{\mathbf{g}}$ lies in $\mathcal{G}$ during the decryption process. This is already taken care of by the test in step 9 of the decryption algorithm.

5 A variant of ACE-Encrypt

In this section, we present a variant of the ACE-Encrypt submission. Several changes were made to the original submission, so that the resulting scheme fits into our uniform framework. This variant is a key encapsulation mechanism that we call ACE-Encrypt′.

5.1 Key Generation

A fully specified group

$$Group = (\mathcal{H}, \mathcal{G}, \mu, \nu, \mathcal{E}, \mathcal{D}, \mathcal{E}', \mathcal{D}')$$

is chosen, together with a random generator $\mathbf{g}_1$ of the group $\mathcal{G}$. The order μ of $\mathcal{G}$ must be prime. We emphasize that the generator $\mathbf{g}_1$ should be chosen at random — this is necessary for the validity of the security theorem.

An additional parameter, *CofactorMode*, must be specified. This parameter takes the value 0 or 1. Here are the rules which should be obeyed in setting this parameter.

- If $\nu = 1$, then *CofactorMode* should be 0.
- If $\nu > 1$, *CofactorMode* may be set to 1 provided $\gcd(\mu, \nu) = 1$.

In addition to *Group*, a hash function *Hash* (see §2.8) and mask generating function *MGF* (see §2.9) must be chosen. It is required that and that $Hash.OutputLen < \log_{256} \mu$.

Since we want *MGF* to be a good entropy smoothing function, one should select either *MGF3* or *MGF4*. As discussed in §2.9, the functions *MGF1* and *MGF2* are not recommended.

Next, numbers $w, x, y, z \in \{0, \ldots, \mu - 1\}$ are chosen at random, and the group elements

$$\mathbf{g}_2 = w \cdot \mathbf{g}_1,\ \mathbf{c} = x \cdot \mathbf{g}_1,\ \mathbf{d} = y \cdot \mathbf{g}_1,\ \mathbf{h} = z \cdot \mathbf{g}_1$$

are computed.

The public key consists of encodings of *Group*, the group elements $\mathbf{g}_1, \mathbf{g}_2, \mathbf{c}, \mathbf{d}, \mathbf{h}$, along with the value *CofactorMode* and an indication of the choice of *Hash* and *MGF*. The precise format of this encoding is not specified here. [Editor's note: should it be?]

The private key consists of the public key, together with the numbers w, x, y, z.

5.2 Encryption

Recall that for a key encapsulation mechanism KEM, the goal is to produce a ciphertext C_0 that is an encryption of a key K, where K is a byte string of length $KeyLen = ACE\_Encrypt'.OutputKeyLen$.

In addition to the recipient's public key, the encryption algorithm takes an optional *format* argument. If the encoding function $\mathcal{E}$ supports multiple formats (as in the elliptic curve case), this formatting argument will be passed to the encoding function.

The encryption scheme works as follows.

1. Choose $r \in \{0, \ldots, \mu - 1\}$.

2. If $CofactorMode = 1$, set $r' = r \cdot \nu \bmod \mu$; otherwise, set $r' = r$.

3. Compute group elements
$$\mathbf{u}_1 = r \cdot \mathbf{g}_1,\ \mathbf{u}_2 = r' \cdot \mathbf{g}_2,\ \tilde{\mathbf{h}} = r' \cdot \mathbf{h}.$$

4. Compute the byte strings
$$EU1 = \mathcal{E}(\mathbf{u}_1, format),\ EU2 = \mathcal{E}(\mathbf{u}_2, format).$$

5. Compute the number
$$\alpha = OS2IP(Hash.eval(EU1 \,\|\, EU2)).$$

6. Compute the number
$$r'' = \alpha \cdot r' \bmod \mu.$$

7. Compute the group element
$$\mathbf{v} = r' \cdot \mathbf{c} + r'' \cdot \mathbf{d}.$$

8. Output the ciphertext
$$C_0 = EU1 \,\|\, EU2 \,\|\, \mathcal{E}'(\mathbf{v})$$
and the key
$$K = MGF(EU1 \,\|\, \mathcal{E}'(\tilde{\mathbf{h}}), KeyLen).$$

5.3 Decryption

The decryption algorithm takes as input a ciphertext C_0 along with the private key.

1. Parse the ciphertext as $EU1 \,\|\, EU2 \,\|\, PEV$, where $EU1$ encodes the group element $\mathbf{u}_1$, $EU2$ encodes the group element $\mathbf{u}_2$, and PEV is a valid partial encoding of a group element. If this parsing step fails, or if $EU1$ and $EU2$ are not encoded using the same *format*[2], then fail.

[2]Version 1.0 of this document failed to include the check that $EU1$ and $EU2$ are encoded using the same *format*; while this check is not necessary for the basic proof of security in §5.4, it is necessary for the reduction in §5.5.2 and the random oracle analysis in §5.5.1.

2. If *CofactorMode* $= 1$, set $\hat{\mathbf{u}}_1 = \nu \cdot \mathbf{u}_1$; otherwise, set $\hat{\mathbf{u}}_1 = \mathbf{u}_1$.

3. If *CofactorMode* $\neq 1$ and $\nu > 1$, test if $\mu \cdot \mathbf{u}_1 = \mathbf{0}$. If not, then fail.

4. Compute the number

$$\alpha = \mathit{OS2IP}(\mathit{Hash.eval}(\mathit{EU1} \,\|\, \mathit{EU2}))$$

5. Compute the number

$$t = x + y\alpha \bmod \mu.$$

6. Test if

$$w \cdot \hat{\mathbf{u}}_1 = \mathbf{u}_2 \text{ and } \mathcal{E}'(t\hat{\mathbf{u}}_1) = \mathit{PEV}.$$

   If not, then fail.[3]

7. Compute the group element

$$\tilde{\mathbf{h}} = z \cdot \hat{\mathbf{u}}_1.$$

8. Output the key

$$K = \mathit{MGF}(\mathit{EU1} \,\|\, \mathcal{E}'(\tilde{\mathbf{h}}), \mathit{KeyLen}).$$

5.4 Security considerations

This scheme differs in only very minor ways from schemes that have been rigorously analyzed in the literature. It most closely resembles the variation of the Cramer-Shoup scheme discussed in detail in [Sho00b].

The security of the scheme is based on the DDH (see §2.11.2), and a few other *specific* assumptions about the hash and mask generating functions. The security reduction is quite tight. One can easily verify the following, using following the line of reasoning in [CS98] and [Sho00b].

$$\begin{aligned} \mathit{Advantage}_{\mathit{ACE\_Encrypt'}}(A) \;= O(\; & \mathit{Advantage}_{\mathit{DDH}}(A_1) + \\ & \mathit{Advantage}_{\mathit{Hash}}(A_2) + \\ & \mathit{Advantage}_{\mathit{MGF}}(A_3) + \\ & q_D \cdot \mu^{-1} \;), \end{aligned}$$

where:

- A_1, A_2, A_3 denote adversaries that run in time essentially the same as A.

- $\mathit{Advantage}_{\mathit{DDH}}$ is as defined in §2.11.2.

[3] For security reasons, one should always perform all of the computations in this step; otherwise, some "timing" information could be gained by the adversary that is not available to it in the formal proof of security.

- $Advantage_{Hash}(A)$ denotes the probability that an adversary A, given encodings $EU1^*$ and $EU2^*$ of two independent, random elements in $\mathcal{G}$, can find encodings $EU1$ and $EU2$ of elements in $\mathcal{G}$, such that $(EU1, EU2) \neq (EU1^*, EU2^*)$, but

  $$Hash.eval(EU1 \parallel EU2) = Hash.eval(EU1^* \parallel EU2^*).$$

  If the group supports multiple encodings, the adversary can choose the format it wants when $EU1^*$ and $EU2^*$ are generated; furthermore, the adversary may choose to use the same or different formats in its choice of $EU1$ and $EU2$; however, $EU1^*$ and $EU2^*$ must be encoded using the same format, and the same holds for $EU1$ and $EU2$.

  If $CofactorMode = 1$, then the adversary may choose $EU1$ to be an encoding of an element of $\mathcal{H}$ that does not necessarily lie in $\mathcal{G}$.

  Note that this problem is a second-preimage collision problem, which is generally believed to be a much harder problem to solve than the problem of finding an arbitrary pair of colliding inputs.

- $Advantage_{MGF}(A)$ denotes the advantage that an adversary A has in distinguishing between the following two distributions. Let $\mathbf{u}_1$ and $\tilde{\mathbf{h}}$ be independent, random elements of $\mathcal{G}$, and let $EU1$ be an encoding of $\mathbf{u}_1$. Let R be a random byte string of length $KeyLen$. The first distribution is $(R, EU1)$, and the second is $(MGF(EU1 \parallel \mathcal{E}'(\tilde{\mathbf{h}}), KeyLen), EU1)$.

- q_D bounds the number of decryption oracle queries made by the adversary A.

The "O" above represents a very small constant, which we have not computed exactly.

5.5 Further remarks

5.5.1 Random oracles and interactive assumptions

We emphasize that this scheme can be proved secure under reasonable intractability assumptions, without resorting to either the random oracle heuristic, and without using "interactive" intractability assumptions as in done in [ABR98].

We stress that a proof of security in the random oracle model *is not* a proof with "just another assumption." One is not assuming a hash function is a random function, since this assumption is patently false. The random oracle model is a *heuristic*, and a proof of security in the random oracle model does not directly imply anything about the security of a system "in the real world."

We also stress that interactive intractability assumptions, like in [ABR98], are qualitatively much stronger than standard intractability assumptions. Indeed, it can be argued that the main activity of theoretical cryptography is to show that breaking a cryptosystem via some kind of subtle, interactive attack is at least as hard as solving some specific, non-interactive problem.

ACE-Encrypt′ can also be proved secure in the random oracle model under the CDH assumption (see [Sho00b]), although the reduction is not nearly as tight as for PSEC-2′. Indeed, the tightness of the reduction for PSEC-2′ and the efficiency of PSEC-2′ are the main reasons for including PSEC-2′ in this proposal.

5.5.2 ACE-Encrypt$'$ and ECIES$'$

One should also note that ACE-Encrypt$'$ is no less secure than ECIES$'$ in a very strong sense. Indeed, assuming the two cryptosystems use the same parameters, then one can show that any adversary A that breaks ACE-Encrypt$'$ can be converted into an adversary A' with about the same running time that breaks ECIES$'$ with the same advantage.

To see this, consider an ECIES$'$ public key $(\mathbf{g}, \mathbf{h})$. Upon obtaining this public key, A' generates w, x, y at random modulo μ, and then computes the ACE-Encrypt$'$ public key

$$(\mathbf{g}_1, \mathbf{g}_2, \mathbf{c}, \mathbf{d}, \mathbf{h}) = (\mathbf{g}, w\mathbf{g}, x\mathbf{g}, y\mathbf{g}, \mathbf{h}).$$

A' then gives this public key to the adversary A.

Now, whenever the adversary A makes a decryption oracle query, then knowing w, x, y, A' performs the validity test of ACE-Encrypt$'$, and if this passes, it uses the decryption oracle of ECIES$'$ to obtain the decrypted symmetric key, giving this to A.

When A invokes the encryption oracle for ACE-Encrypt$'$, A' invokes the encryption oracle for ECIES$'$, obtaining an encoding of a group element $\mathbf{u}_1^*$. Then using w, x, y, A' easily constructs the remaining components of a corresponding ACE-Encrypt$'$ ciphertext, and gives this to A.

One needs to check that A' carries out a legal chosen ciphertext attack, i.e., that A' never attempts to submit the target ciphertext to the decryption oracle subsequent to the invocation of the encryption oracle. But this follows easily from the following claim: for any two *valid* ACE-Encrypt$'$ ciphertexts $C_0 = \mathit{EU1} \parallel \mathit{EU2} \parallel \mathit{PEV}$ and $C_0^+ = \mathit{EU1}^+ \parallel \mathit{EU2}^+ \parallel \mathit{PEV}^+$, if $\mathit{EU1} = \mathit{EU1}^+$, then $C_0 = C_0^+$. This claim relies on the fact that the validity test for a ciphertext C_0 as above ensures that $\mathit{EU1}$ and $\mathit{EU2}$ are encoded using the same format. If this were not done, then simply by replacing $\mathit{EU2}$ by a different encoding $\mathit{EU2}^+$ of the same group element, one would violate the above claim.

When A terminates and outputs a bit $\hat{b}$, A' also terminates and outputs the same thing.

It is easily seen that this simulation is perfect, and that whatever advantage A has in breaking ACE-Encrypt$'$, A' has the same advantage in breaking ECIES$'$.

5.5.3 ACE-Encrypt and ACE-Encrypt$'$

We outline the major differences between ACE-Encrypt and ACE-Encrypt$'$.

- We have generalized the algorithm to work with an arbitrary, abstract group, and to work with an arbitrary message authentication code and symmetric key encryption scheme.

- We have chosen not to use the rather specialized universal one-way hash function to compute the quantity α. Instead, we use a standard cryptographic hash, and make a specific — but reasonable — "second preimage collision resistance" assumption.

  The proposed standard need not necessarily preclude the possibility of using such a specialized hash function, so long as we allow such a hash to have a variable length key that is stored in the public key.

- We have chosen not to use the rather specialized entropy-smoothing hash function to derive the key K. Instead, we again use a standard cryptographic hash, and make a specific — but again, reasonable — "entropy smoothing" assumption.

  The proposed standard need not necessarily preclude the possibility of using such a specialized hash function, so long as we allow such a hash to have a variable length key that is stored in the public key.

6 RSA-OAEP

6.1 Message encoding functions

EME-OAEP is a fully specified version of Bellare and Rogaway's original OAEP scheme for message encoding [BR94].

In general, a message encoding scheme *EME* of this type specifies two algorithms:

- $EME.Encode(M, L, ELen)$ takes as input a message M and a label L, and an output length $ELen$. Here, M and L are byte strings whose lengths are bounded, as described below. It outputs a byte string E of length $ELen$.

- $EME.Decode(E, L)$ takes as input a byte string E and a label L. It attempts to find a message M such that $EME.Encode(M, L, |E|) = E$. It returns M if such an M exists, and otherwise fails.

In addition to this, the mechanism should specify a bound $EME.Bound$ such that when $EME.Encode(M, L, ELen)$ is invoked, the condition $|M| \le ELen - EME.Bound$ should hold; if not, the encoding algorithm fails. Additionally, the encoding algorithm may also fail if $|L|$ exceeds some (very large) implementation-defined bound.

The algorithm $EME.Encode$ will in general be probabilistic, so that the same message can be encoded in a number of ways.

6.2 EME-OAEP

We now describe EME-OAEP.

The scheme is parameterized by a hash function *Hash* (see §2.8) and a mask generation function *MGF* (see §2.9). Current standards, as well as the RSA-OAEP submission to ISO, recommend the use of the function *MGF1* using *Hash*. Let $HLen = Hash.OutputLen$.

The quantity $EME\_OAEP.Bound$ is defined as

$$EME\_OAEP.Bound = 2 \cdot HLen + 1.$$

6.2.1 Encoding function

The algorithm $EME\_OAEP.Encode(M, L, ELen)$ runs as follows:

1. Check that $|M| <= ELen - 2 \cdot HLen - 1$; if not, then fail.

2. Generate a random byte string r of length $HLen$.

3. Let pad be the byte string of length $ELen - |M| - 2 \cdot HLen$ consisting of a sequence of 0-bytes, followed by a single 1-byte.

4. Set $x = Hash.eval(L) \,\|\, pad \,\|\, M$.

5. Set $s = MGF(r, ELen - HLen) \oplus x$.

6. Set $t = MGF(s, HLen) \oplus r$.

7. Output $E = t \,\|\, s$.

6.2.2 Decoding function

The algorithm $EME\_OAEP.Decode(E, L)$ runs as follows.

1. Let $ELen = |E|$.

2. Check if $ELen \geq 2 \cdot HLen + 1$; if not, then fail.

3. Parse E as $E = t \,\|\, s$, where $|t| = HLen$ and $|s| = ELen - HLen$.

4. Set $r = MGF(s, HLen) \oplus t$.

5. Set $x = MGF(r, ELen - HLen) \oplus s$.

6. Test that x is of the form $x = Hash.eval(L) \,\|\, pad \,\|\, M$, where pad is a byte string consisting of zero or more 0-bytes, followed by a 1-byte. If not, then fail.

7. Output M.

6.3 RSA-OAEP

We describe a generic RSA encryption scheme, based on an arbitrary message encoding mechanism EME. If one uses EME-OAEP, the resulting scheme is called RSA-OAEP.

6.3.1 Key generation

The public key consists of an RSA modulus n that is the product of two large primes, and an exponent e. It also specifies any parameters of EME (such as $Hash$ and MGF, in the case of EME-OAEP). Let $nLen$ denote the length, in bytes, of n.

The secret key consists of the decryption exponent d, where $ed \equiv 1 \bmod n$.

6.3.2 Encryption

The algorithm to encrypt a message M with label L runs as follows.

1. Set $E = \mathit{EME.Encode}(M, L, \mathit{nLen} - 1)$.
2. Set $m = \mathit{OS2IP}(E)$.
3. Set $c = m^e \bmod n$.
4. Output $C = \mathit{I2OSP}(c, \mathit{nLen})$.

6.3.3 Decryption

The algorithm to decrypt a ciphertext C with label L runs as follows.

1. If $|C| \neq \mathit{nLen}$, then fail.
2. Let $c = \mathit{OS2IP}(C)$.
3. Check that $c \leq n - 1$; if not, then fail.
4. Set $m = c^d \bmod n$.
5. Set $E = \mathit{I2OSP}(m, \mathit{nLen} - 1)$; note that this step may fail if m is too large.
6. Set $M = \mathit{EME.Decode}(E, L)$; note that this step may fail.

It should be stressed that any error codes returned by any subroutines called by the decryption algorithm should all be converted to a unique error code — in general, the precise cause of the error should not be revealed.

6.4 Defects of RSA-OAEP

RSA-OAEP suffers from two defects.

The first is a security defect. It was a widely held belief that the general OAEP construction was secure against adaptive chosen ciphertext attack, assuming the underlying trapdoor permutation was one-way. This belief is based on a supposed random-oracle proof in [BR94]. This of course would imply the security of RSA-OAEP in the random oracle model, assuming that RSA is one-way. However, it was recently shown in [Sho00a] that the proof of security of the general OAEP construction was invalid, and further, the general construction can not be proven secure using standard proof techniques.

This result by itself does not imply that RSA-OAEP is insecure; it simply invalidates the original justification of its security. In fact, in [Sho00a], it is shown that RSA-OAEP with $e = 3$ is secure (in the random oracle model). This result is extended by [FOPS00] to arbitrary e. It should be noted however, that the security reduction is much less efficient in [FOPS00] than that proposed in [BR94] for OAEP.

The fact that RSA-OAEP can be proved secure is essentially an accident. The proofs of security exploit particular algebraic properties of the RSA function.

In [Sho00a], a slight variant of OAEP is presented, called OAEP+. A detailed proof of security is given, on the general assumption of a trapdoor one-way permutation. Moreover, the security reduction is much more efficient than that of [FOPS00] or even [BR94].

Another defect of RSA-OAEP is that it only encrypts messages of a bounded length. Because of this, RSA-OAEP is really only useful as a key encapsulation mechanism (see §2.2), and it is left to application engineers to implement a "digital envelope" for encrypting longer messages. See Remark 1 for a discussion about why we believe that this standard should provide a complete solution to the "digital envelope" problem, rather than just a partial solution. Also, using RSA-OAEP for nothing more than key encapsulation completely wastes one of the main feature of OAEP, namely, its very good "message expansion" rate. Indeed, if all one wants to do with RSA is encapsulate a key, then one is better served using the Simple RSA scheme in §8, as that method is both simpler and quantitatively more secure.

Because of these two defects, we propose that the new ISO standard contain a variation of RSA-OAEP+ that offers both a higher level of security than RSA-OAEP, while at the same time introduces a standard for encrypting messages of arbitrary length using RSA.

It is a question for debate as to whether the ISO standard should contain RSA-OAEP at all, given the above defects. This is a point for discussion.

7 RSA-OAEP+

In this section, we propose a new encryption scheme, called RSA-OAEP+. It has better provable security properties than RSA-OAEP, and also provides a secure mechanism for encrypting messages of arbitrary length.

7.1 Extended message encoding functions

To facilitate encryption of arbitrary length cleartexts, we extend the notion of a message encoding scheme.

In general, an extended message encoding scheme *XEME* specifies two algorithms:

- $\mathit{XEME}.\mathit{Encode}(M, L, \mathit{ELen}, \mathit{KeyLen})$ takes as input a message M, a label L, an encoding output length ELen, and a key output length KeyLen. Here, M and L are byte strings whose lengths are bounded, as described below. It outputs a pair (E, K) of byte strings with $|E| = \mathit{ELen}$ and $|K| = \mathit{KeyLen}$.

- $\mathit{XEME}.\mathit{Decode}(E, L, \mathit{KeyLen})$ takes as input a byte string E and a label L. It attempts to find a message M and a key K such that $\mathit{EME}.\mathit{Encode}(M, L, |E|, \mathit{KeyLen}) = (E, K)$. It returns the pair (M, K) if it exists, and otherwise fails.

In addition to this, the mechanism should specify a bound $\mathit{XEME}.\mathit{Bound}$ such that when $\mathit{XEME}.\mathit{Encode}(M, L, \mathit{ELen}, \mathit{KeyLen})$ is invoked, the condition $|M| \leq \mathit{ELen} - \mathit{XEME}.\mathit{Bound}$ should hold; if not, the encoding algorithm fails. Additionally, the encoding algorithm may also fail if $|L|$ or KeyLen exceed some (very large) implementation-defined bound.

The algorithm *XEME.Encode* will in general be probabilistic, so that the same message can be encoded in a number of ways.

7.2 XEME-OAEP+

We now describe the extended message encoding scheme XEME-OAEP+.

The scheme is parameterized by a mask generation function *MGF* (see §2.9) and an integer $MaskLen \geq 1$. Any of the functions described in §2.9 are suitable.

The quantity *XEME_OAEP+.Bound* is defined as

$$XEME\_OAEP{+}.Bound = 2 \cdot MaskLen + 1.$$

Let $(I0, I1, \ldots)$ denote the values $(I2OSP(0,4), I2OSP(1,4), \ldots)$.

7.2.1 Encoding function

The algorithm $XEME\_OAEP{+}.Encode(M, L, ELen, KeyLen)$ runs as follows.

1. Check that $|M| <= ELen - 2 \cdot MaskLen - 1$; if not, then fail.

2. Generate a random byte string r of length *MaskLen*.

3. Let *pad* be the byte string of length $ELen - |M| - 2 \cdot MaskLen$ consisting of a sequence of 0-bytes, followed by a single 1-byte.

4. Set $x = pad \,\|\, M$.

5. Set
$$check = MGF(I0 \,\|\, r \,\|\, x \,\|\, I2OSP(KeyLen, 4) \,\|\, L, MaskLen).$$

6. Set
$$x' = MGF(I1 \,\|\, r, ELen - 2 \cdot MaskLen) \oplus x.$$

7. Set
$$s = check \,\|\, x'.$$

8. Set
$$t = MGF(I2 \,\|\, s, MaskLen) \oplus r.$$

9. Output
$$E = t \,\|\, s$$
and
$$K = MGF(I3 \,\|\, r, KeyLen).$$

7.2.2 Decoding function

The algorithm $XEME\_OAEP+.Decode(E, L)$ runs as follows.

1. Let $ELen = |E|$.

2. Check if $ELen \geq 2 \cdot MaskLen + 1$; if not, then fail.

3. Parse E as $E = t \,\|\, s$, where $|t| = MaskLen$ and $|s| = ELen - MaskLen$.

4. Set
$$r = MGF(I2 \,\|\, s, MaskLen) \oplus t.$$

5. Parse s as $check \,\|\, x'$, where $|check| = MaskLen$ and $|x'| = ELen - 2 \cdot MaskLen$.

6. Set
$$x = MGF(I1 \,\|\, r, ELen - 2 \cdot MaskLen) \oplus x'.$$

7. Test if x is of the form $x = pad \,\|\, M$, where pad is a byte string consisting of zero or more 0-bytes, followed by a 1-byte; if not, then fail.

8. Test if
$$check = MGF(I0 \,\|\, r \,\|\, x \,\|\, I2OSP(KeyLen, 4) \,\|\, L, MaskLen).$$
If not, then fail.

9. Output M and
$$K = MGF(I3 \,\|\, r, KeyLen).$$

Remark 11 This encoding scheme is very similar to that of [Sho00a]. Besides a few inconsequential formatting changes, this scheme deals with a label L and produces a key K of length $KeyLen$. The scheme in [Sho00a] does not deal with labels or key outputs at all. Notice that both $KeyLen$ and L are hashed into the value $check$ — this is important for the security of the scheme. $\triangleleft$

Remark 12 In general, we have kept the changes between EME-OAEP and XEME-OAEP+ minimal. But since some changes were anyway necessary, we took the liberty to propose a couple of further changes.

The main change is that we use the function MGF in several places, and we insert the strings $I0, I1$, etc., into the different invocations of MGF. This is done so that these can be more properly modeled as *independent* random oracles, as required in the proof of security. $\triangleleft$

7.3 RSA-OAEP+

We describe a generic *extended* RSA encryption scheme that uses an arbitrary *extended* message encoding scheme *XEME*. If the XEME-OAEP+ encoding scheme is used, the resulting encryption scheme is called RSA-OAEP+. We call this an extended RSA encryption scheme, since it handles messages of arbitrary length.

This scheme also makes use of a symmetric key encryption scheme *SKE* (see §2.5) and a one-time message authentication code *MAC* (see §2.6). The techniques we use are similar to those for building a hybrid encryption scheme (see §2.7).

7.3.1 Key generation

Just as for RSA-OAEP, the public key consists of an RSA modulus n that is the product of two large primes, and an exponent e. It also specifies any parameters of *XEME*. Let *nLen* denote the length, in bytes, of n.

The secret key consists of the decryption exponent d, where $ed \equiv 1 \bmod n$.

7.3.2 Encrypting short messages

To encrypt a message M with label L, where $|M| \leq nLen - XEME.Bound - 1$, one does the following.

1. Set $(E, K) = XEME.Encode(M, L, nLen - 1, 0)$; note that K is the empty string.
2. Set $m = OS2IP(E)$.
3. Set $c = m^e \bmod n$.
4. Output $C = I2OSP(c, nLen)$.

7.3.3 Decrypting short messages

To decrypt a ciphertext C with label L, where $|C| \leq nLen$, one does the following.

1. If $|C| < nLen$, then fail.
2. Let $c = OS2IP(C)$.
3. Check that $c \leq n - 1$; if not, then fail.
4. Set $m = c^d \bmod n$.
5. Set $E = I2OSP(m, nLen - 1)$; note that this step may fail if m is too large.
6. Set $(M, K) = XEME.Decode(E, L, 0)$; note that this step may fail, and also that K is the empty string.
7. Output M.

7.3.4 Encrypting long messages

To encrypt a message M with label L, where $|M| > nLen - XEME.Bound - 1$, one does the following.

1. Let $M = M_0 \,\|\, M_1$, where $|M_0| = nLen - XEME.Bound - 1$.
2. Set $(E, K) = XEME.Encode(M_0, L, nLen - 1, SKE.KeyLen + MAC.KeyLen)$.
3. Set $m = OS2IP(E)$.
4. Set $c = m^e \bmod n$.
5. Set $C_0 = I2OSP(c, nLen)$.
6. Parse K as $K = k \,\|\, k'$, where $|k| = SKE.KeyLen$ and $|k'| = MAC.KeyLen$.
7. Encrypt M_1 under the key k using SKE, and let C_1 be the resulting ciphertext.
8. Apply the MAC to C_1 using the key k', obtaining tag.
9. Output the ciphertext $C = C_0 \,\|\, C_1 \,\|\, tag$.

7.3.5 Decrypting long messages

To decrypt a ciphertext C with label L, where $|C| > nLen$, one does the following.

1. Test if $|C| > nLen + MAC.TagLen$; if not, then fail.
2. Parse C as $C = C_0 \,\|\, C_1 \,\|\, tag$, where $|C_0| = nLen$ and $|tag| = MAC.TagLen$.
3. Let $c = OS2IP(C_0)$.
4. Check that $c \le n - 1$; if not, then fail.
5. Set $m = c^d \bmod n$.
6. Set $E = I2OSP(m, nLen - 1)$; note that this step may fail if m is too large.
7. Set $(M_0, K) = XEME.Decode(E, L, SKE.KeyLen + MAC.KeyLen)$. Note that this step may fail.
8. Test if $|M_0| = nLen - XEME.Bound - 1$; if not, then fail.
9. Parse K as $K = k \,\|\, k'$, where $|k| = SKE.KeyLen$ and $|k'| = MAC.KeyLen$.
10. Apply the MAC to C_1 using the key k', and check if the resulting tag matches the given tag; if not, then fail.
11. Decrypt C_1 under the key k using SKE, and let M_1 be the resulting plaintext.
12. Output $M = M_0 \,\|\, M_1$.

7.4 Security considerations

It is straightforward to adapt the proof of security in [Sho00a] to show that this scheme is secure in the random oracle model against adaptive chosen ciphertext attack, assuming the RSA inversion problem is hard.

That proof implies that for any adversary A, its advantage in breaking the cryptosystem RSA_OAEP+ is bounded by

$$\begin{aligned} Advantage_{RSA\_OAEP+}(A) \;=\; O(\; & Advantage_{RSA}(A_1) \; + \\ & Advantage_{PRBG}(A_2, l_1) \; + \\ & q_D \cdot Advantage_{MAC}(A_3, l_1, l_2) \; + \\ & q_D q_H \cdot 2^{-MaskLen} \;) \end{aligned}$$

Here,

- A_1 is an algorithm that runs in time roughly equivalent to that of A, plus $O(q_H^2)$ applications of the RSA function,
- A_2, A_3 are adversaries whose running times are about the same as A,
- $Advantage_{RSA}(A)$ denotes the success probability of an algorithm A has in solving a random instance of the RSA inversion problem,
- q_D is a bound on the number of decryption oracle queries made by A,
- q_H is a bound on the number of random oracle queries made by A,
- l_1 is a bound on the length of the target cleartext, and
- l_2 is a bound on the length of ciphertexts submitted to the decryption oracle.

Note that this security reduction is actually somewhat more efficient than the original (and incorrect) security reduction for RSA-OAEP in [BR94]. It is also far more efficient than the security reduction in [FOPS00]. In that reduction, the algorithm A' for inverting RSA is somewhat slower than that of RSA-OAEP+, but worse, if the advantage of A is ϵ, then the success probability of A' is about ϵ^2.

Even though the security reduction for RSA-OAEP+ is tighter than that for RSA-OAEP, we should perhaps point out that because of the term $O(q_H^2)$ in the running time of the RSA inversion algorithm, this reduction actually says very little about the security of, say, 1024-bit RSA. This is because one can (most likely) factor 1024-bit numbers in less time than that required by the implied RSA inversion algorithm. However, as pointed out in [Sho00a], for exponent $e = 3$, there is a much more efficient security reduction whose running time is linear in q_H. Is this a reason recommend the use of $e = 3$? Perhaps. Alternatively, one can use the Simple RSA scheme (see §8).

Of course, if the security reduction for RSA-OAEP+ implies very little about concrete security, the security reduction for RSA-OAEP in [FOPS00] says even less.

8 Simple RSA

We also suggest for possible inclusion in the ISO standard the following very simple version of RSA. It is based on the ideas in [BR93].

The scheme we present is a key encapsulation mechanism (see §2.2), which can be turned into an encryption scheme as described in §2.7.

The main advantages of this scheme are its simplicity and the fact that it yields a much more efficient (and hence meaningful) security reduction compared to that for OAEP or OAEP+. The disadvantage is that ciphertexts are a little bit larger.

8.1 Key Generation

Just as for RSA-OAEP, the public key consists of an RSA modulus n that is the product of two large primes, and an exponent e. It also specifies a mask generation function MGF (see §2.9). Let $nLen$ denote the length, in bytes, of n.

The secret key consists of the decryption exponent d, where $ed \equiv 1 \bmod n$.

8.2 Encryption

Recall that Simple RSA is a key encapsulation mechanism, and so the goal of the encryption algorithm is simply to produce a pseudo-random key K of length $KeyLen = SimpleRSA.OutputKeyLen$ and a ciphertext C that encrypts K.

The encryption algorithm runs as follows.

1. Generate a random number $r \in \{0, \ldots, n-1\}$.
2. Compute $y = r^e \bmod n$.
3. Compute $K = MGF(I2OSP(r, nLen), KeyLen)$.
4. Compute $C = I2OSP(y, nLen)$.
5. Output the ciphertext C and the key K.

8.3 Decryption

Given a ciphertext C, decryption runs as follows.

1. Check that $|C| = nLen$; if not, then fail.
2. Set $y = OS2IP(C)$.
3. Check that $y < n$; if not, then fail.
4. Compute $r = y^d \bmod n$.
5. Compute $K = MGF(I2OSP(r, nLen), KeyLen)$.
6. Output the key K.

8.4 Security considerations

The security of Simple RSA can be analyzed in the random oracle model in a manner very similar to that in [BR93], where we model the invocation of *MGF* as a random oracle query. It is easy to show that

$$Advantage_{SimpleRSA}(A) \leq Advantage_{RSA}(A') + nBound/q_D, \tag{8}$$

where

- A' is an algorithm for solving a random instance of the RSA problem that runs in time roughly the same as that of A; more precisely, the running time is that of A, plus the time to perform q_H exponentiations modulo n, where q_H is a bound on the number of random oracle queries made by A;
- q_D is a bound on the number of decryption oracle queries made by A;
- *nBound* is an *lower bound* on n.

We sketch a proof of this. Let $\mathbf{G}_0$ be the original attack game played by adversary A, and let S_0 be the event that A correctly guesses the hidden bit b in game $\mathbf{G}_0$. Let H denote the random oracle mapping elements of $\mathbf{Z}_n$ to bit strings of length *KeyLen*. Let $y^* \in \mathbf{Z}_n$ denote the target ciphertext, and let $r^* = (y^*)^{1/e} \in \mathbf{Z}_n$.

We next define a game $\mathbf{G}_1$ that is the same as game $\mathbf{G}_0$, except that if the target ciphertext y^* was submitted to the decryption oracle prior to the invocation of the encryption oracle, then the game is halted. Let S_1 be the event in game $\mathbf{G}_1$ corresponding to the event S_0.

Let F_1 be the event that game $\mathbf{G}_1$ is halted as above. Clearly, $\Pr[F_1] \leq nBound/q_D$, and since games $\mathbf{G}_0$ and $\mathbf{G}_1$ proceed identically until F_1 occurs, it follows by Lemma 1 that $|\Pr[S_0] - \Pr[S_1]| \leq nBound/q_D$.

We next define a game $\mathbf{G}_2$ that is the same as $\mathbf{G}_1$, except that (1) the target ciphertext is generated at the beginning of the game, and (2) if the adversary ever queries H at r^*, we halt the game. Let S_2 be the event in game $\mathbf{G}_2$ corresponding to the event S_0.

It is clear by construction that $\Pr[S_2] = 1/2$, since the key $H(r^*)$ is independent of everything else that is accessible to the adversary in game $\mathbf{G}_2$, either directly or indirectly. Indeed, only the encryption oracle evaluates H at r^* in this game.

Let F_2 be the event that game $\mathbf{G}_2$ is halted as above. It is clear that both games $\mathbf{G}_1$ and $\mathbf{G}_2$ proceed identically until F_2 occurs, and so by Lemma 1, we have $|\Pr[S_1] - \Pr[S_2]| \leq \Pr[F_2]$. Thus, it suffices to bound $\Pr[F_2]$.

We claim that

$$\Pr[F_2] \leq Advantage_{RSA}(A')$$

for an algorithm A' that runs in time bounded as described above. The inequality (8) will follow immediately.

Algorithm A' runs as follows. It takes as input a random RSA modulus n, an RSA exponent e, and a random element $y^* \in \mathbf{Z}_n$. It creates a public key using N and e, and then lets adversary A run in game $\mathbf{G}_2$.

When adversary A invokes the encryption oracle, algorithm A' responds to A with the pair (K^*, y^*), where K^* is a random bit string of length *KeyLen*, and y^* is the above-mentioned input to A.

Algorithm A' simulates the random oracle H as well as the decryption oracle, as follows. For every input $r \in \mathbf{Z}_n$ to the random oracle, A' computes $y = r^e \in \mathbf{Z}_n$, and places the triple consisting of r, y, and the random value $K = H(r)$ in a table; however, if $y = y^*$, algorithm A' instead outputs r and halts. When the adversary A submits a ciphertext $y \in \mathbf{Z}_n$ to the decryption oracle, algorithm A' looks up the value y in the above table to determine if the random oracle has been evaluated at $r = y^{1/e} \in \mathbf{Z}_n$. If so, algorithm A' responds to the decryption oracle invocation with the value $K = H(r)$ stored in the table. Otherwise, algorithm A' generates a fresh random key K, and places the pair (y, K) in a second table; moreover, if in the future the adversary A should evaluate the random oracle at a point $r \in \mathbf{Z}_n$ such that $r^e = y$, then the key K generated above will be used for the value of $H(r)$.

It is clear that algorithm A' perfectly simulates the view of A, and that A' outputs a solution to the given instance of the RSA problem with probability equal to $\Pr[F_2]$.

That completes the proof of security.

Quantitatively, it is clear that Simple RSA provides a much better security reduction than RSA-OAEP+ (or RSA-OAEP). This advantage becomes even more pronounced when one analyzes the security of *many* cleartexts encrypted under a single public key (as formally modeled in [BBM00]). In this setting, one can exploit the well-known random self-reducibility property of the RSA inversion problem to easily show that the security of the Simple RSA key encapsulation mechanism does not degrade at all as the number of ciphertexts increases. Note that this argument will be valid only if the number r in the encryption algorithm for Simple RSA is chosen uniformly modulo n, or at least with a distribution that is computationally indistinguishable from the uniform distribution.

For RSA-OAEP+, the security degrades linearly with the number of ciphertexts, since one cannot use the random self-reducibility property, and must instead use a "hybrid argument." The reason the random self-reducibility property cannot be used is that in RSA-OAEP+ (like RSA-OAEP) the ciphertext is not uniformly distributed modulo n.

References

[ABR98] M. Abdalla, M. Bellare, and P. Rogaway. DHAES: an encryption scheme based on the Diffie-Hellma problem. Submission to IEEE P1363, 1998.

[BBM00] M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: security proofs and improvements. In *Advances in Cryptology–Eurocrypt 2000*, 2000.

[BDPR98] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology–Crypto '98*, pages 26–45, 1998.

[BLK00] J. Baek, B. Lee, and K. Kim. Secure length-saving ElGamal encryption under the computational Diffie-Hellman assumption. In *Proc. 5th Australian Conference on Information, Security, and Privacy*, 2000.

[Bon98] D. Boneh. The Decision Diffie-Hellman Problem. In *Ants-III*, pages 48–63, 1998. Springer LNCS 1423.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR94] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology—Eurocrypt '94*, pages 92–111, 1994.

[CGH98] R. Canetti, O. Goldreich, and S. Halevi. The random oracle model, revisted. In *30th Annual ACM Symposium on Theory of Computing*, 1998.

[CS98] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology–Crypto '98*, pages 13–25, 1998.

[DDN91] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *23rd Annual ACM Symposium on Theory of Computing*, pages 542–552, 1991.

[DDN98] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography, 1998. Manuscript (updated, full length version of STOC paper).

[FO99] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology–Crypto '99*, pages 537–554, 1999.

[FOPS00] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is still alive! Cryptology ePrint Archive, Report 2000/061, 2000. `http://eprint.iacr.org`.

[JM96] D. Johnson and S. Matya. Asymmetric encryption: evolution and enhancements. *Cryptobytes*, 2(1), 1996. `http://www.rsasecurity.com/rsalabs`.

[JN01] A. Joux and K. Nguyen. Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. Cryptology ePrint Archive, Report 2001/003, 2001. `http://eprint.iacr.org`.

[Luc00] S. Lucks. The sum of PRPs is a secure PRF. In *Advances in Cryptology–Eurocrypt 2000*, 2000.

[NR97] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th Annual Symposium on Foundations of Computer Science*, 1997.

[OP01] T. Okamoto and D. Pointcheval. The gap-problems: a new class of problems for the security of cryptographic schemes. In *Proc. 2001 International Workshop on Practice and Theory in Public Key Cryptography (PKC 2001)*, 2001.

[RS91] C. Rackoff and D. Simon. Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology–Crypto '91*, pages 433–444, 1991.

[Sho97] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology–Eurocrypt '97*, 1997.

[Sho00a] V. Shoup. OAEP reconsidered. Cryptology ePrint Archive, Report 2000/060, 2000. `http://eprint.iacr.org`.

[Sho00b] V. Shoup. Using hash functions as a hedge against chosen ciphertext attack. In *Advances in Cryptology–Eurocrypt 2000*, 2000.

[Sta96] M. Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology–Eurocrypt '96*, pages 190–199, 1996.